SANDIA REPORT

SAND2012-4667
Unlimited Release
Printed June, 2012

Summary of Work for ASC L2 Milestone 4465: Characterize the Role of the Mini-Application in Predicting Key Performance Characteristics of Real Applications

Richard F. Barrett, Paul S. Crozier, Douglas W. Doerfler, Simon D. Hammond, Michael A. Heroux, Paul T. Lin, Heidi K. Thornquist, Timothy G. Trucano, Courtenay T. Vaughan

Center for Computing Research
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1319

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2012-4667
Unlimited Release
Printed June, 2012

Summary of Work for ASC L2 Milestone 4465: Characterize the Role of the Mini-Application in Predicting Key Performance Characteristics of Real Applications

A suite of "mini-apps" has been created to serve as proxies for full scale applications. Each miniapp is designed to represent a key performance characteristic that does or is expected to significantly impact the runtime performance of a scientific or engineering application program. In this paper we introduce a methodology for assessing the ability of these mini-apps to represent these performance issues, and apply the methodology to a set of applications.

Acknowledgment

The Mantevo project began as a Sandia laboratory funded project in 2007.

Support for this work was provided through the Advanced Simulation and Computing (ASC) program funded by U.S. Department of Energy's National Nuclear Security Agency. This effort was greatly enhanced by interactions with staff throughout Sandia as well as many external organizations. It is heartening to discover the active interests, supported by broad and deep expertise, of the computational science community.

Contents

List of Figures

List of Tables

Summary

A suite of "mini-apps" has been created to serve as proxies for full scale applications. Each miniapp is designed to represent a key performance characteristic that does or is expected to significantly impact the runtime performance of a scientific or engineering application program. In this paper we introduce a methodology for assessing the ability of these mini-apps to represent these performance issues, and apply the methodology to a set of applications.

The work reported herein is in support of the ASC Level 2 milestone, "Characterize the Role of the Mini-Application in Predicting Key Performance Characteristics of Real Applications". The deliverable for the milestone was a presentation, 23 May 2012. The review committee is

- James A. Ang, Scalable Architectures (1422) manager
- Teddy D. Blacker, Simulation Modeling Sciences (1543) manager
- Robert J. Hoekstra (lead), Scalable Algorithms (1424) manager
- Allen L. McPherson, Computer Scientist, Los Alamos National Laboratory
- William J. Rider, Computational Physicist, Computational Shock and Multiphysics (1443)
- Charles (Bert) Still, Computational Physicist, Lawrence Livermore National Laboratory

The statement of the milestone is:

The Mantevo project includes a set of application proxies, referred to as mini-apps, and designed by code developers to represent key runtime performance characteristics of their applications. SNL will analyze two of these mini-apps to determine how well they represent the full application programs. Specifically, SNL will profile the runtime performance of the mini-app and application, characterizing the relationship between the two on at least two HPC platforms (including Cielo).

Chapter 1

Introduction

Throughout its history, computational science has grown in conjunction with the computational capabilities provided to it [24, 32, 12, 31]. New computational capabilities inspire new solutions methods, the time to solutions of existing methods may be decreased, and more complex problems can be made possible. However, effectively understanding those computational capabilities typically requires access to them.

We have entered a period where the previous approach to increasing computational capability has played out. Computer architects have therefore fundamentally altered their approach, and it is widely recognized that current computational characteristics must change.

For example, the scientific community is currently investing significant effort in preparing for, influencing, and directing the development of an exascale computing capability in an accelerated manner [36, 2]. In order to have an influence on the architectures that could be produced within the accelerated time frame, it is critical to provide meaningful and actionable information throughout the codesign space. Exploring the the performance characteristics and capabilities of full scale scientific and engineering application computer programs is prohibitive, and probably intractable on platforms that don't yet exist.

In recognition of the various challenges in providing an effective environment for this scale of computing, the United States Department of Energy's Advanced Simulation and Computing (ASC[1]) Campaign's has acquired a breadth of important testbed systems. These testbeds provide a concrete means for investigating, understanding, and influencing... Yet porting full application codes to these testbeds is a prohibitive labor intensive effort. The Mantevo project [18] has produced a set of proxies for these applications, called miniapps, that enable rapid exploration of key performance issues that impact a broad set of scientific application programs of interest to the ASC program as well as a broader community.

Yet how can we be sure that these miniapps adequately represent that which they are intended to represent? The key contribution of the work described herein is a methodology, rooted in formal verification and validation (V&V) efforts that have been developed for experimental science, for determining the quality of the miniapp as it pertains to a large, complex application code.

[1] `http://nnsa.energy.gov/asc/supercomputers/`

Chapter 2

Overview of the Mantevo Project

The Mantevo project [18], initiated as a Sandia LDRD, was motivated by some questions arising from the Trilinos project [17]. These questions concerned the direction of some coding implementations targeting emerging and expected future architectures, including multi-core, many-core, and GPU-accelerated high performance computers. The goal was to create a set of application- relevant codes enabling rapid exploration of algorithmic options and their mapping to computing platforms.

As discussed above, each miniapp is designed to focus attention on a key performance characteristic of one or more application programs, enabling agile exploration of a variety of issues that impact performance, ranging from low level hardware capabilities through the codesign space [14] to the application.

Miniapps are designed to be a tool, useful throughout the co-design space. For example, each of the ASC Exascale working groups[1] have identified miniapps for use from hardware registers to the application. Also, the Exascale Grand Challenge (XGC) project at Sandia National Laboratories in includes a 3DI efforts are targeted at heterogeneous integration of Logic (MPU Process), DRAM (DRAM Process), and Silicon Photonics (Custom Process). Unlike a benchmark, the result of which is a value to be ranked, the output of a miniapp is information, which must be interpreted within some often subjective context. Unlike a compact application, which is designed to capture some sort of physics behavior, miniapps are designed to capture some key performance issue in the full application. Unlike a skeleton application, which is designed for only focusing on inter-process communication perhaps involving a "fake" computation, miniapps create a meaningful context in which to explore the key performance issue. Miniapps are developed and owned by application code teams. Miniapps are intended to by modified, and thus are limited to $O(1k)$ source lines of code (SLOC), allowing for unconstrained modification. Once no longer useful for these purposes, a miniapps life will end. Miniapps are freely available as open source software under an LGPL license.

The current set of miniapps in the Mantevo project are listed in Table 2.1. The first miniapp was HPCCG, which formed and solved a sparse linear system of equations. Although this has provided an important capability, it was soon realized that in order to provide a stronger tie to applications of interest, the context in which the linear system is formed

[1] `https://asc.llnl.gov/exascale/`

Miniapp	*Description*
HPCCG	Sparse linear system solver (CG)
miniFE	Implicit finite element method (FEM)
miniGhost	Finite difference or volume method
miniMD	Molecular dynamics
miniITCFE	Implicit Thermal Conduction (FEM)
miniETCFE	Explicit Dynamics (FEM)
miniXyce	Circuit simulation

Table 2.1. List of Mantevo miniapps

needed strengthened. The result was miniFE, putting the linear system into the context of an implicit finite element solver.

Chapter 3

Methodology

Miniapps are designed to provide a predictive capability for some key performance issue in a full application. Ensuring that a miniapp completely fulfills its intent is a difficult and probably ongoing task. Further, the runtime behavior complex scientific application is typically problem dependent, and therefore its important to understand the different ways that a code can be used and have a means for configuring the miniapp to mimic the important features under consideration.

Thus our approach is to build up a "body of evidence" in support of the goals of a miniapp, combining formal verification and validation (V&V) techniques with our knowledge and experience bases.

Verification is the process of determining that a model implementation accurately represents the developers conceptual description of the model and the solution to the model. *Validation* is the process of determining the degree to which a model is an accurate representation of the "real world" (in this case the performance characteristics of the "real" application) from the perspective of the intended uses of the model[1]. That is, within the context of the intent of the comparisons of a model with the "real world", we must verify that the applications ("real world") and miniapps ("model") compare well in the performance dimensions of interest. This is our defined means of assessing that the miniapps are accomplishing what they are designed and required to do (or not). All of the work (and possibly art) in this methodology will be defining a set of comparisons that allow us to draw conclusions of this kind about the miniapps. We must also understand how close these comparisons should be for us to be able to conclude that the miniapps are suitably accurate models of real code performance, or that they aren't. There will clearly be significant components of judgment embedded in this methodology given the difficult nature of

[1]These terms are as defined by the American Society of Mechanical Engineers (ASME, 2006) and the American Institute of Aeronautics and Astronautics (AIAA, 1998)), and this usage has basically been adopted by the United States Departments of Energy (DOE) and Defense (DoD). IEEE definitions (IEEE, 1991) are also useful and relevant in this context. These are: *Verification* – "The process of evaluating a (software) system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. (2) Formal proof of program correctness." Validation – "The process of evaluating a (software) system or component during or at the end of the development process to determine whether it satisfies specified requirements." Succinctly and informally, and compatible with these more formal definitions, the reader can assume that we mean the following: *Verification* – the process of assessing the evidence that a miniapp is correctly implemented. *Validation* – the process of assessing the evidence of how closely the miniapp resembles the full application in the performance domain of interest.

this problem, but the goal is to achieve some minimal level (at least) of objective evidence that informs us constructively about the fidelity of the miniapps. This approach requires extensive knowledge of, and experience developing, executing, profiling, maintaining, and extending multi-scale, multi-physics scientific and engineering application software, targeting highest performance computing platforms. It also requires a strong understanding of the miniapps and their intended use: what they are intended to represent and what they are not intended to represent. We combine this knowledge into a formal verification and validation (V&V) methodology that lets us examine experimental and predicted data.

This methodology adheres to the spirit of experimental validation as described in [27, 26, 28, 38] because validation referents are intended to be representative of the empirical (that is, "real") performance of the full applications. However, its important to note that this V&V process is executed within the goals of miniapps, and thus we are not concerned with ensuring any sort of V&V in regard to the application goals such as correctness of the algorithms and output. In other words, in considering the performance fidelity of minapps, we are not addressing questions about the verification and validation of the full application; we are assuming that this has been done or, if not, that this is an issue that is not relevant to the immediate goals of developing miniapps. The status of this assumption is in fact of interest, but beyond our immediate scope. Our focus is strictly on the computational runtime characteristics. Beyond this issue we also stress that miniapps are not intended to reproduce specific physics and mathematics represented by the full application. To some degree, we therefore have an operating assumption that a valid miniapp can approximate the runtime performance characteristics of a full application to a useful degree without reproducing the mathematics and physics of the full application to a useful degree. This may be an assumption that is worthy of fuller consideration also, but once again it is beyond the scope of our near term priorities.

3.1 Verification

Mantevo miniapps have been configured so that they produce some outcome that is measurable with some level of confidence of correctness. For example, miniFE solves a Partial Differential Equation (PDE) such that the residual norm of the linear system solution is within an acceptable tolerance. MiniGhost includes a means for ensuring correctness.

With regard to the application, we necessarily begin with the assumption it meets it V&V requirements. That said, our methodology still includes a strong element of software verification. In particular, because our validation approach is designed to expose differences in the runtime characteristics between the miniapp and application, the differences seen can point to areas where both the miniapp and application should be examined. In some sense, then, this is a code-to-code V&V exercise, whereby lack of agreement between the two can strengthen the (always ongoing) V&V efforts of each. This of course is not a definitive result, since both codes may be similarly incorrect, and therefore somewhat controversial. However, given the assumptions stated above, this sort of information can still provide a useful service

if properly understood.

3.2 Validation

For a set of diagnostic runtime performance characteristics or elements, which we loosely refer to as the *performance domain*,

$$\{D\} = D_1, D_2, \ldots, D_n, \tag{3.1}$$

let

$$\{B\} = B_1, B_2, \ldots, B_n, \tag{3.2}$$

be a corresponding set of baseline full application observational referents, (the "validation data") and let

$$\{A\} = A_1, A_2, \ldots, A_n, \tag{3.3}$$

be a set of corresponding miniapp measurements.

We then consider the difference between the application referents and the miniapp measurements in the performance domain defined by (3.1) as some kind of mathematical norm, which we will also call a *validation metric*:

$$X_i = \|B_i - A_i\|_i, \forall i. \tag{3.4}$$

Here, we have suggested that the difference measurement – the norm – might vary for each component of the performance domain. Clearly, this can become extremely complex and examples presented below will help clarify this. A very simple example could be a situation in which a positive number specifies every component of performance domain. In such a case, we then could simply have:

$$X_i = \|B_i - A_i\|_i = |B_i - A_i|, \forall i. \tag{3.5}$$

But one component of the performance domain might be specified this way, while another component might be as general as a functional or a time series, in which case the norm is far more complex than simply measuring the absolute value of the difference between two numbers. There is also the possibility that stochastic characteristics must be attached to one or more of the performance dimensions, which further complicates the mathematics

that might underlie the validation metric. We do assume that all the components of the performance domain can have a norm distance definition attached to them. In something as extremely complicated as overall runtime performance behavior, even this assumption might fail if we had to deal with qualitative factors in performance. We do not see the need for this level of generality for this discussion.

The point of introducing a validation metrics that measures the difference between miniapp performance and the application referent is to draw some conclusion about how well the miniapp is reproducing the performance behavior of the full application. A simple illustration of this logic is as follows. Suppose that the validity of the miniapp was determined by how accurately it reproduces the performance of the full application in the performance domain. Then the differences in Equation 3.4 provide the means for assessing validity. Thus, given the measured set of values $X_i, \text{for} i = 1, \ldots, n$, suppose "valid" accuracy can be assessed using a set of threshold accuracies $T_i^1, T_i^2, \text{for} i = 1, \ldots, n$. Then assessment of the validation metric information might then be posed as:

$$V_i = \begin{cases} \text{predictive}, & \text{for } T_i^1 \leq X_i \leq T_i^2 \\ \text{caution}, & \text{for } T_i^2 \leq X_i \leq T_i^3 \\ \text{not predictive}, & \text{for } X_i \geq T_i^3 \end{cases} \tag{3.6}$$

where V_i is a validity statement attached to performance domain dimension i for some threshholds T_i^j, for $j = 1, \ldots, 3$.

While Equation 3.6 looks like a generally useful algorithm for assessment, we caution there is a great deal of overloading going on in this simple expression. For example, the choice of thresholds could clearly be extremely difficult. The willingness to even evaluate validity based on a relatively direct threshold assessment is open to debate. And, developing the set $V_i, i = 1, \ldots, n$ leaves open the issue of how all of this information is combined into a single appraisal of the validity of the minapp. Nonetheless, this logic is a clear illustration of the kind of ideal thinking that should underlies the validation assessment of miniapps.

Choice of diagnostics defining the performance domain is clearly a challenge. Example diagnostics for the set D include inter-process communication, which can then be further categorized, such as the number and distribution of partners, and the size and frequency of message traffic. Example observational referents B for the full application may be empirically measured, which we denote as B^M. Or the referents may need to be forecast or estimated using expert judgment, perhaps with a tool like the SSTK, denoted B^p. Using referents that are not empirically measured for validation is sometimes called face validation. While accepted by several communities, including DoD, clearly *face validation* has weaker validation inference associated with it than the use directly observed empirical referents. We expect that we might have to use a mixture of both in the validation methodology for miniapps. In general, we emphasize that measurements involving miniapps A must be carefully designed and captured. We also emphasize that having confidence in A and B requires having accumulated meaningful verification evidence.

This framework provides direct advantages. First, the input information D, B, and A and are open to challenge and refinement, are mutable and extensible, and thus the role interpretive judgment in the final results of validity assessment is transparent within the context of use. For example, new diagnostics, new or corrected baseline observations, and new or corrected measurements could be added to the model in the service of better assessment. Second, the way the results are computed is can be easily subjected to peer-review scrutiny.

3.3 A Discussion of Metrics

The choice of measure significantly influences the way in which differences (the validation metric $\|\cdot\|_i$) between validation data B_i and miniapp measurements A_i are viewed. It will be important to careful choose and explore options...*(Expecting to engage Scott Mitchell w.r.t. his LDRD work in this area.)*

For our validation work, we choose metrics based on their ability to emphasize or highlight differences between the application characteristic under consideration and the miniapp measurements.

Discussion of multiple experiments as a means of defining bounding error bars. Attributable to computer system issues such as ...

Chapter 4

Experimental Platforms

We employed a variety of computing environments in our work, including the current ASC capability machine (Cielo) and two capacity machines (Red Sky and TLCC2 Chama). Table 4.1 lists the speeds and feeds. Representative processors found in workstations let us examine some particular issues of interest at that scale.

4.1 Cielo: Cray XE6

Cielo, an instantiation of a Cray XE6, is composed of AMD Opteron Magny-Cours processors, connected using a Cray custom interconnect named Gemini, and a light-weight kernel (LWK) operating system called Compute Node Linux. The system, illustrated in Figure 4.1, consists of 8,944 compute nodes, for a total of 143,104 cores.

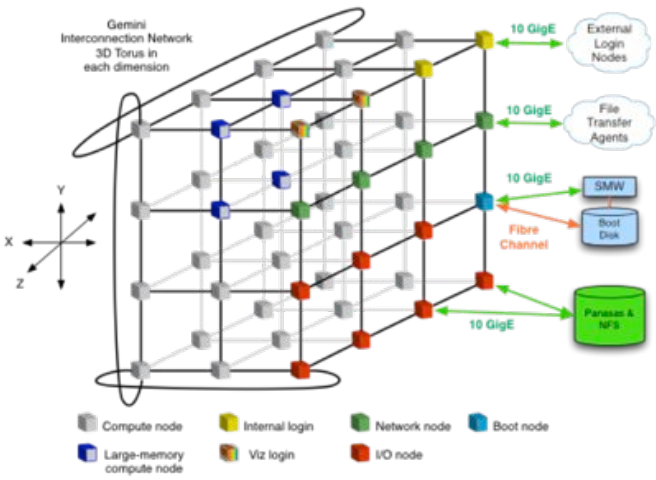


Figure 4.1. Cielo XE6 architecture. Image courtesy of Cray Inc.

	Chama	*Cielo*	*Red Sky*
Vendor	Appro	Cray	Sun/Oracle
Installed	2012	2011	2008
OS	TOSS	CNL	TOSS
# Compute Nodes	1,232	8,894	8,894
Sockets/Node	2	2	2
Cores/Socket	8	8	4
Total Cores	19,712	142,304	18,544
Processor	Intel Sandy Bridge	AMD Magny-Cours	Intel Nehalem
Frequency (GHz)	2.6	2.4	2.93
FLOPS/Clock	8	4	4
GFLOPS/Node	332.8	153.6	93.76
Memory Type	1600 MHz DDR3	1333 MHz DDR3	1333 MHz DDR3
Cache L1	8×32 KB I, D	8×64 KB, I,D	4×32 KB, I,D
Cache L2	8×256 KB	8×512 KB	4×256 KB
Cache L3	20 MB	12 MB (10MB)	8 MB
Memory/Node (GB)	32	32	12
Mem BW/Node (GB/s)	102	85.3	64.0
NUMA Regions/Node	2	4	2
Network Interface	Qlogic QDR IB	Cray Gemini	Mellanox QDR IB
Network Topology	Fat-Tree, 3-level	3-D Torus	3-D torus
PingPong Latency (μs)		1.3	
Bi-dir Inj. BW/node (GB/s)	5	10	
Bi-dir Link Bandwidth (GB/s)	8	9.4, 12-bit links 18.8, 24-bit links	

Table 4.1. Comparison of Chama, Cielo and Red Sky

Each Cielo node consists of two oct-core AMD Opteron Magny-Cours processors[1]. Each Magny-Cours processor is divided into two memory regions, called NUMA nodes, each consisting of four processor cores (illustrated in Figure 4.2). Thus each compute node consists of 16 processor cores, evenly divided among four NUMA nodes, which are connected using HyperTransport[2] version 3. The links between NUMA nodes run at 6.4 GigaTransfers per second (GT/s). Each core has a dedicated 64 kByte L1 data cache, a 64 kByte L1 instruction cache, and a 512 kByte L2 data cache, and the cores within a NUMA node share a 6 MByte L3 cache (of which 5 MBytes are user available).

Cielo compute nodes are connected using Cray's Gemini 3-D torus high-speed interconnect, illustrated in Figure 4.3. A Gemini ASIC supports two compute nodes. The X and Z

[1]Magny-Cours processors are also available with 12 cores divided into 6-core NUMA nodes, which form the basis of the new Hopper II computer at NERSC (`http://www.nersc.gov/nusers/systems/hopper2/`).

[2]`http://www.hypertransport.org`

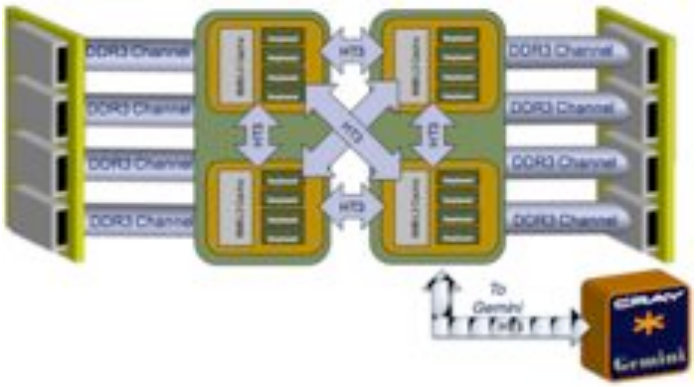
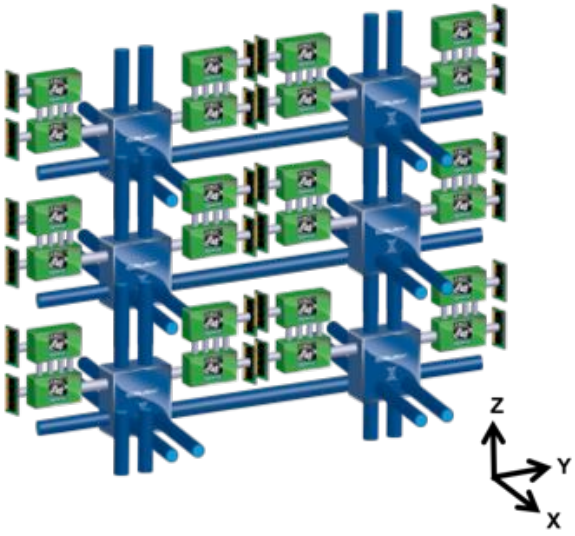
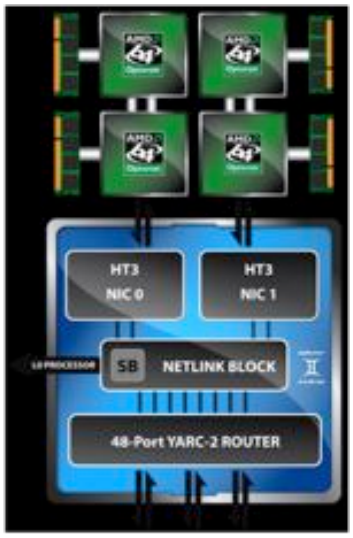


Figure 4.2. The XE6 compute node architecture. Images courtesy of Cray, Inc.



(a) Cray XE6 Gemini Architecture



(b) XE6 Node Architecture with AMD Magny-Cours processors and Cray Gemini high-speed interconnect

Figure 4.3. The XE6 Gemini architecture. Images courtesy of Cray, Inc.

dimensions use twice as many links as the Y dimension (24 bits and 12 bits respectively) and introduces an asymmetry to the nodes in terms of bandwidth in the torus. This needs to be taken into account when configuring a system in order to balance the bisection bandwidth of each dimensional slice in the torus. Cielo is configured as an $18 \times 8 \times 24$ 3-D torus. Injection bandwidth is limited by the speed of the Opteron to Gemini HyperTransport link, which runs at 4.4 GT/s. Links in the X and Z dimensions have a peak bi-directional bandwidth of 18.75 GB/s, and the Y dimension peaks at 9.375 GB/s.

Relative to its SeaStar predecessor used in the Cray XT series [7], Gemini provides an improvement to the achievable asymptotic bandwidth for point-to-point communication. There are two potential bottlenecks to consider: injection bandwidth and link bandwidth. Injection bandwidth is limited by the speed of the Opteron to Gemini HyperTransport link, which runs at 4.4 GT/s. Link bandwidth is determined by the signaling rate and the width of the link. Due to Gemini's double-density packaging, links in the X and Z dimensions are twice the width of links in the Y dimensions (24-bits vs. 12-bits wide). Gemini is also intended to provide support for fine grained remote load-store-style messaging, as is typical of partitioned global address space (PGAS) languages [35].

4.2 Red Sky

Red Sky is a cluster composed of Sun Vayu blades, two nodes per blade, connected QDR InfiniBand (illustrated in Figure 4.4. Each node consists of dual-sockets fitted with the

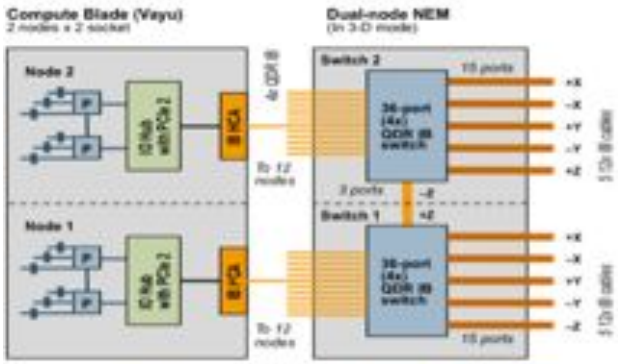


Figure 4.4. Red Sky node and IB interconnect (courtesy of Sun Microsystems)

Intel 5570, "Nehalem-EP" 2.93 GHz quad-core processors, with each processor having an integrated memory controller with three 1333 MHz DDR3 memory channels (6 GB total per socket). Each blade also integrates 10/100 Mbps and Gigabit Ethernet channels for cluster management. The midplane (NEM) integrates 4x QDR IB switches. The Mellanox

HCA connects the nodes to the IB router and has a peak bandwidth of 40 Gbits/sec to the NEM modules. Two 36-port QDR InfiniBand (IB) switches are used per chassis and the port connections for the toroidal interconnect. For each of the 36-port switches (SW) twelve ports are used to connect to the node HCAs, nine to connect SW1 to SW2, and the remaining fifteen ports form the external X, Y, Z links for the 3D torus. Each row, consisting of 12 racks, forms a $6 \times 2 \times 8$ (X,Y,Z) torus building block. The logical $6 \times 6 \times 8$ (X,Y,Z) torus maps to physical $12 \times 3 \times 8$ node configuration. Logical Y dimension "folds over" at the last physical row and the torus is completed in an adjacent rack of physical row 1. Logical X dimension skips every other rack in the physical X dimension. Logical Z dimension is self contained within a rack and is fixed at 8.

Red Sky is a modular system that consists of three sections, identified by the three names: Red Sky, Red Horizon, and Red Mesa, integrated as required to meet programmatic and operational requirements, for a total of 42,400 processor cores. For the purposes of the work herein, a group of racks in 3 rows of cabinets that include 2,823 core nodes (22,584 cores), yielding a peak performance of 265 TFLOPS.

The software environment on Red Sky uses the TOSS 1.3-4 which is based on Red Hat RHEL 5 Linux with several patches. The InfiniBand uses OpenFabrics software configured with the OpenSM Subnet manager incorporating a custom routing engine developed at Sandia for the 3D torus.

4.3 Chama

Chama, the latest Tri-lab Capacity Computer (TLCC) sited at Sandia, is composed of 1,232 compute nodes connected using a Qlogic QDR Infiniband interconnect, and the TOSS operating system[3]. Each node consist of two Intel Xeon Sandy Bridge oct-core processors (illustrated in Figure 4.5(a)), configured as a Intel,Qlogic QDR FAT Tree, illustrated in Figure 4.5(c). Each core has a dedicated 32 kByte L1 data cache, a 32 kByte L1 instruction cache, and a 256 kByte L2 data cache, and the cores within each socket share a 20 MByte L3 cache.

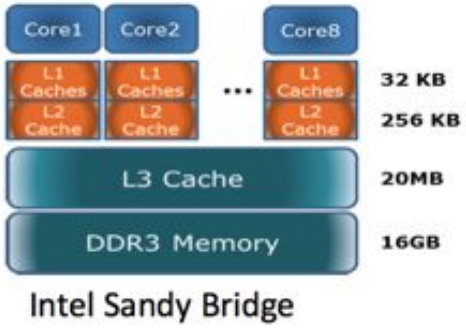
4.4 Some workstations

Workstations allow us to focus in on some particular issues. We used the following:

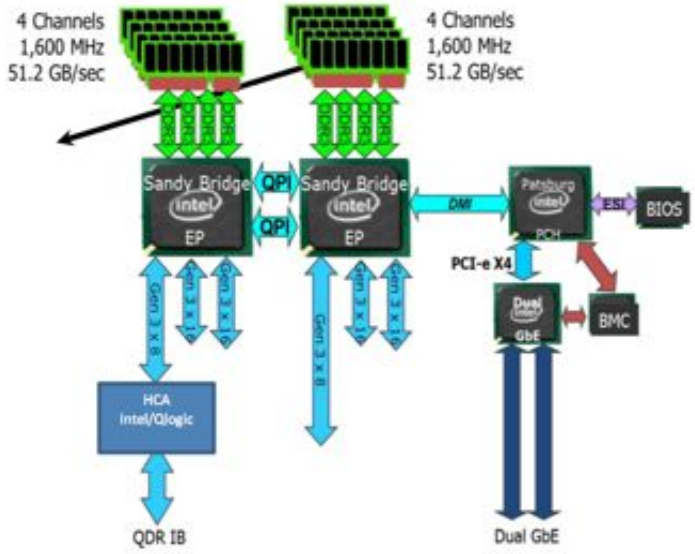
- dual socket quad-core Intel Nehalem 5560@ 2.8 GHz processors.
- dual socket quad-core Intel Nehalem 5570@ 2.93GHz processors.

[3]ASC Program Tripod Operating System Software (TOSS), a Tri-Lab packaging of the CHAOS/SLURM environment.

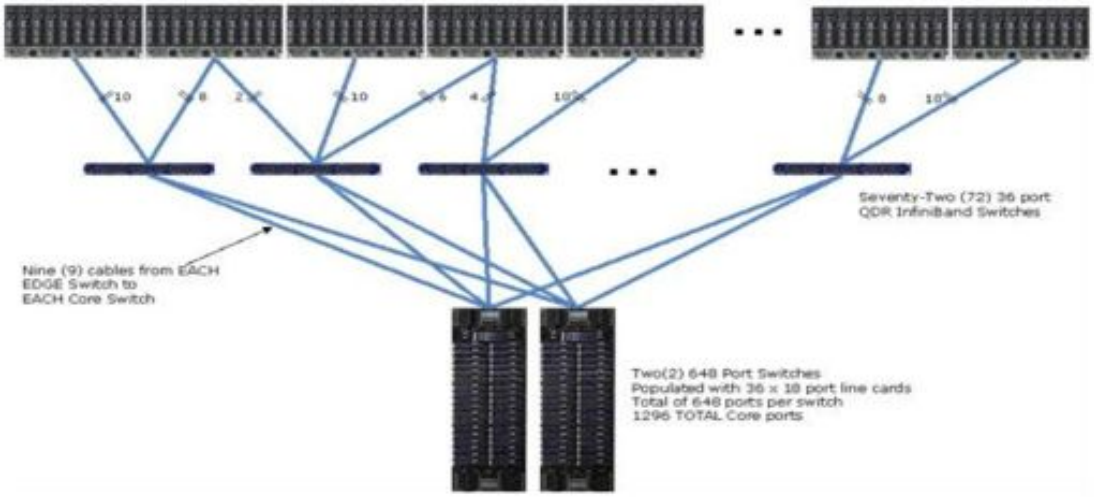
- dual socket oct-core AMD Magny-Cours 6136@2.4 GHz
- dual socket 12-core AMD Magny-Cours @2.1 GHz



(a) Sandy Bridge



(b) Node



(c) Fat tree interconnect

Figure 4.5. Chama architecture

Chapter 5

Making the link to full applications

In this section, we apply our methodology for linking miniapps to problem-dependent application performance factors and evaluating the fidelity of that linkage, then illustrate our approach by examining four miniapps, each in the context of a large application code. Our goal is to begin to answer the question, "Under what conditions does a miniapp represent a key performance characteristic in a full app?" Runtime profiling information is presented in support of well-specified key performance issues for each code, providing some acceptable level of confidence that the miniapp is representative of the their relevant computations, in a manner that will enable experimentation with different programming models and languages, communication mechanisms, and architectures.

MiniMD was developed to model the molecular dynamics Lennard-Jones potential, such as that found in LAMMPS. MiniGhost was designed to provide a means for exploring alternatives to the Bulk-Synchronous Parallel programming model with data aggregation interprocess communication strategy, such as that found in CTH, a shock physics code. MiniFE was developed to examine Krylov-based linear equations solution methods applied within the context of a Krylov solver in an implicit finite element method application on unstructured meshes, such as that used in Charon, a semiconductor device simulator. MiniXyce, currently under development, is intended to represent Xyce, a circuit simulation code.

5.1 A Molecular Dynamics code

LAMMPS, an acronym for Large-scale Atomic/Molecular Massively Parallel Simulator, is a classical molecular dynamics code[1] [29, 30]. It has potentials for soft materials (biomolecules, polymers) and solid-state materials (metals, semiconductors) and coarse-grained or mesoscopic systems. It can be used to model atoms or, more generically, as a parallel particle simulator at the atomic, meso, or continuum scale.

The physical domain is divided into three dimensional boxes, one per parallel process. Each process computes forces on atoms in its box using information from nearby processes (illustrated in Figure 5.1). As they migrate among processes, the atoms carry along their molecular topology. Communication is via nearest-neighbor six-way stencil. Parallel scaling

[1] `http://lammps.sandia.gov/`

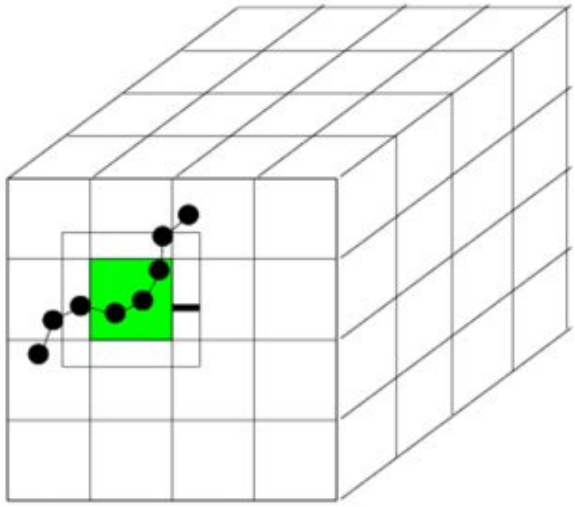


Figure 5.1. Molecular dynamics computation

would be N/P if load is perfectly balanced. Computation scales as N/P, communication scales as $(N/P)^{2/3}$ (for large problems), and memory scales as N/P.

The computational steps for the Velocity-Verlet algorithm are

- update V by 1/2 step (using F),
- update X (using V),
- build neighbor lists (occasionally),
- compute F (using X),
- apply constraints and boundary conditions (on F),
- update V by 1/2 step (using new F), then
- output and diagnostics

Each of N particles is a point mass

- atom
- group of atoms (united atom)
- macro- or meso-particle

Particles interact via empirical force laws

- all physics in energy potential force
- pair-wise forces (LJ, Coulombic)
- many-body forces (EAM, Tersoff, REBO)
- molecular forces (springs, torsions)
- long-range forces (Ewald)

Integrate Newton's equations of motion

- $F = ma$
- set of N, coupled ODEs
- advance as far in time as possible

Properties via time-averaging ensemble snapshots (vs MC sampling).

MiniMD is designed to model the force computations in typical molecular dynamics applications. The algorithms and implementation used closely mimic these same operations as performed in LAMMPS. In particular, miniMD uses the Lennard-Jones potential (Equation 5.2),

$$V_{LJ} = 4\varepsilon\left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right) \tag{5.1}$$

$$= \varepsilon\left(\left(\frac{r_m}{r}\right)^{12} - 2\left(\frac{r_m}{r}\right)^{6}\right) \tag{5.2}$$

where ε is the depth of the potential well, σ is the finite distance at which the inter-particle potential is zero, r is the distance between the particles, and r_m is the distance at which the potential reaches its minimum. At $r_m, V_{LJ} = -\varepsilon$. The distances are related as $r_m = 2^{1/6}\sigma$.

The tasks involved in the parallel processing implementation of this model are:

1. **Construct neighbors list** : Each atom is assigned to a cell in a three-dimensional bin. Using some distribution definition, a neighbor list (array `neigh`) is constructed for each parallel process. (We abbreviate this task as "neigh".)
2. **Collect off-process neighbors** : Point-to-point inter-process communication collects off-process neighbor data on to the owning parallel process. (We abbreviate this task as "comm".)

3. **Compute forces** : Calculate the forces acting upon each atom by the other atoms. (We abbreviate this task as "forces".)

The main focus of miniMD is on the force calculation, with the code segment shown in Figure 5.2.

```
for (i = 0; i < nlocal; i++) {
  neighs = neighbor.firstneigh[i];
  numneigh = neighbor.numneigh[i];
  xtmp = x[i][0];
  ytmp = x[i][1];
  ztmp = x[i][2];
  for (k = 0; k < numneigh; k++) {
    j = neighs[k];
    delx = xtmp - x[j][0];
    dely = ytmp - x[j][1];
    delz = ztmp - x[j][2];
    rsq = delx*delx + dely*dely + delz*delz;
    if (rsq < cutforcesq) {
      sr2 = 1.0/rsq;
      sr6 = sr2*sr2*sr2;
      force = sr6*(sr6-0.5)*sr2;
      f[i][0] += delx*force;
      f[i][1] += dely*force;
      f[i][2] += delz*force;
      f[j][0] -= delx*force;
      f[j][1] -= dely*force;
      f[j][2] -= delz*force;
    }
  }
}
```

Figure 5.2. Code: miniMD force calculation

One easy sanity check diagnostic for molecular dynamics simulations is to make sure that total energy is conserved. For an NVE ensemble molecular dynamics simulation, the total energy (the sum of the kinetic and potential energies) should remain essentially constant throughout the duration of the simulation. If not, either the timestep size is too aggressive, there are discontinuities in the force field, or there are bugs in the simulation software. For long-time simulations, there should be little (or no) discernible drift in the total energy.

Given a second molecular dynamics simulation package, additional diagnostics can be performed. In our case, we can compare miniMD output against LAMMPS output. Given

the same input parameters (timestep size, force field definition, density, initial temperature), very similar steady state average thermodynamic properties (kinetic energy, potential energy, pressure) should be observed for both codes.

In fact, good energy conservation is observed for both miniMD and LAMMPS, and good thermodynamic property agreement is observed between the two codes (illustrated in Figure 5.3).

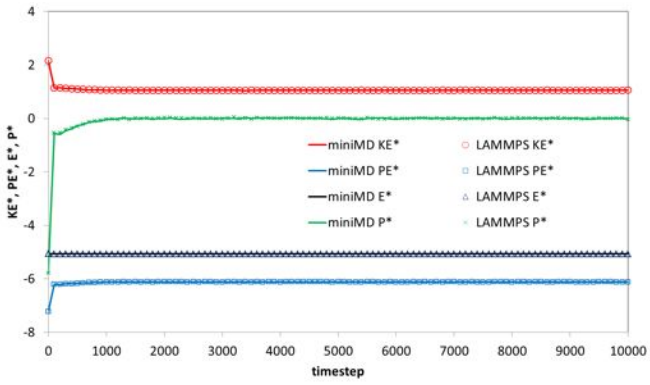


Figure 5.3. Thermodynamic properties output for miniMD and LAMMPS. Good energy conservation and good agreement are observed for this simple NVE ensemble test problem consisting of 32,000 LJ atoms at a reduced density of 0.8442.

5.1.1 Model Abstractions

MiniMD and LAMMPS(LJ) are quite similar, providing a strong candidate experiment for testing our methodology. That said, miniMD does not include the additional complex logic present in the LAMMPS implementation, resulting in shorter runtimes.

LJ is one particular type of atomic interaction model that can be used in molecular dynamics. In the future, miniMD may be expanded to include more complex atomic interaction models in order to perform an expanded set of comparisons with LAMMPS.

5.1.2 Performance Domain

The strong connection of miniMD to LAMMPS(LJ) provides a rather straightforward means of applying our methodology. The diagnostics are the time to solution for each computational phase, using four different input decks, varying the number of atoms in the simulation.

D_1 : Total time
D_2 : Force calculation time
D_3 : Time for construction of neighbor list
D_4 : Time for inter-process communication

For each, the input deck was configured as

- $\rho^* = 0.8442$
- $T* = 1.444$
- Timestep* = 0.00462
- 4000, 8000, 16000, and 32000 atoms
- 1000 time steps

5.1.3 Diagnostic: Total time

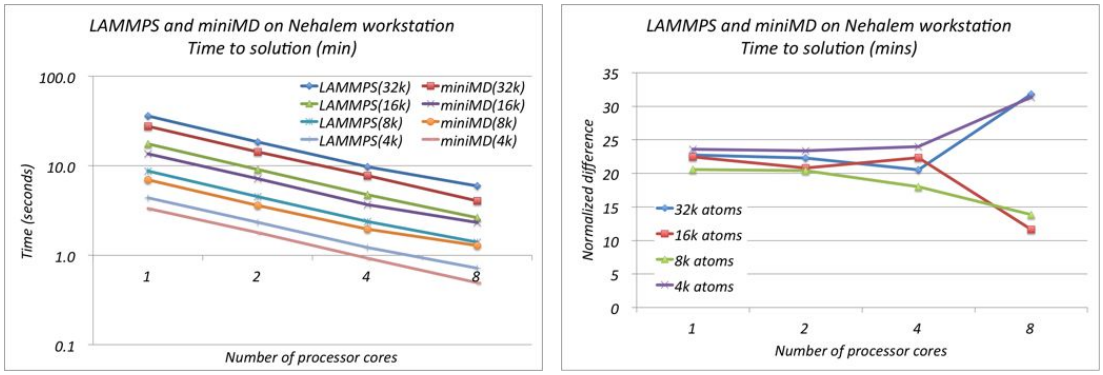
Performance results on a Nehalem workstations and Muzia are illustrated in Figure 5.1.3. These are the minimum times of three trials for each phase.

5.1.4 Diagnostic: Force calculation time

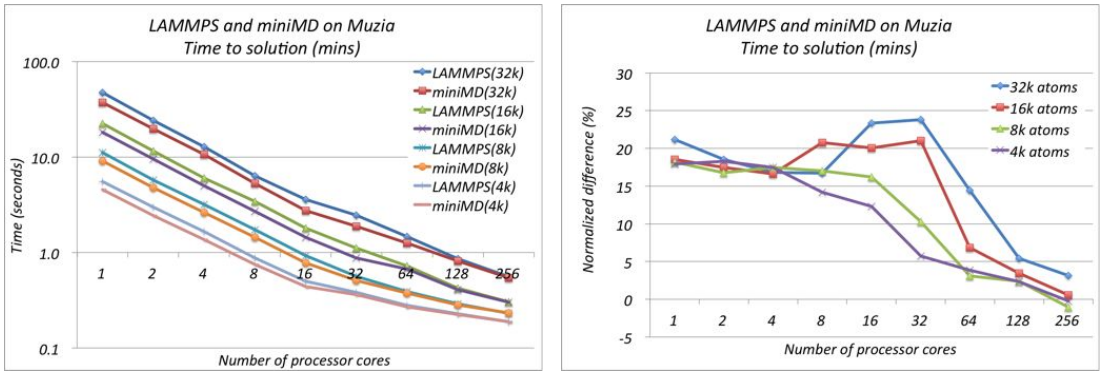
Performance results on a Nehalem workstations and Muzia are illustrated in Figure 5.5. These are the minimum times of three trials for each phase. The graphs on the left are direct comparisons, in terms of time. The graphs on the right show the results of our validation methodology. Here we use the normalized form $X_i = (B_i - A_i)/B_i$, converted to a percentage.

5.1.5 Diagnostic: Time for construction of neighbors

Performance results on a Nehalem workstations and Muzia are illustrated in Figure 5.6. These are the minimum times of three trials for each phase. The graphs on the left are direct comparisons, in terms of time. The graphs on the right show the results of our validation methodology. Here we use the normalized form $X_i = (B_i - A_i)/B_i$, converted to a percentage.



(a) Time



(b) Proportional difference

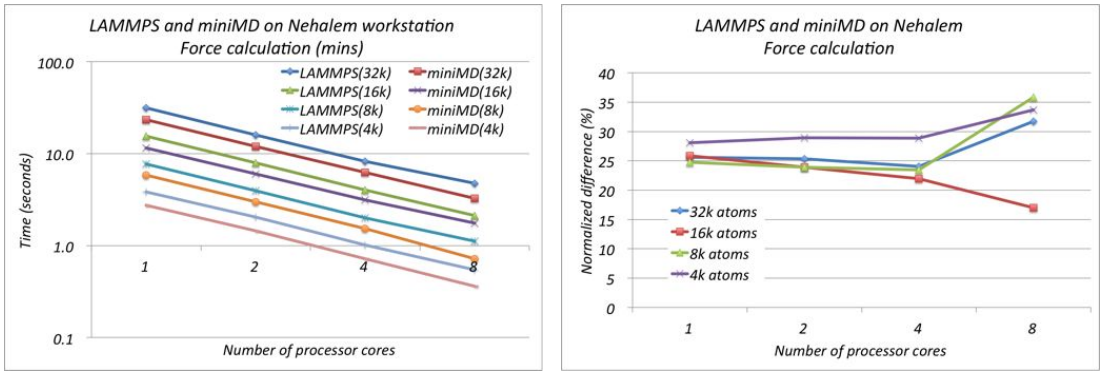
Figure 5.4. Performance, LAMMPS and miniMD : Total time, strong scaling

5.1.6 Diagnostic: Time for inter-process communication

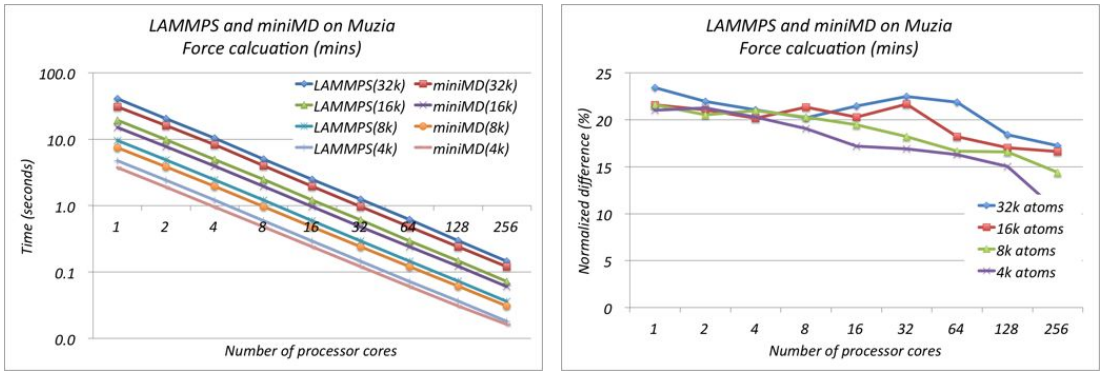
Performance results on a Nehalem workstations and Muzia are illustrated in Figure 5.7. These are the minimum times of three trials for each phase. The graphs on the left are direct comparisons, in terms of time. The graphs on the right show the results of our validation methodology. Here we use the normalized form $X_i = (B_i - A_i)/B_i$, converted to a percentage.

5.1.7 Summary

These are the minimum times of three trials for each phase. The average standard deviations are shown in Table 5.1.



(a) Force time



(b) Proportional difference

Figure 5.5. Performance, LAMMPS and miniMD : Force time, strong scaling

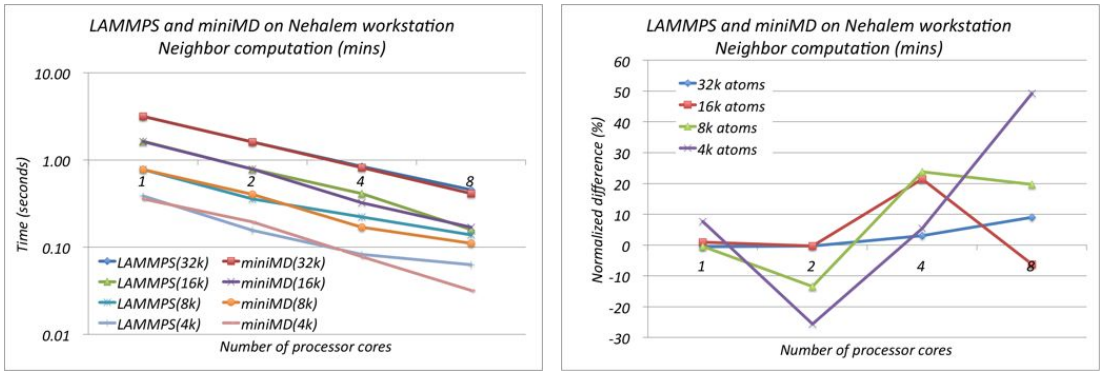
5.1.8 Performance on Red Sky

Strong and weak scaling experiments were run on Red Sky. Performance is illustrated in Figure 5.8.

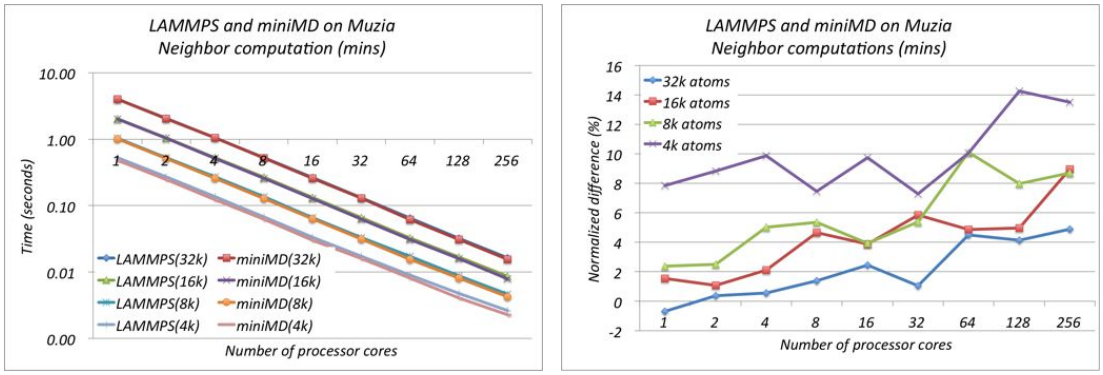
5.1.9 Discussion

Although miniMD closely mimics the Lennart-Jones algorithm in LAMMPS, it does so within the context of a single, focused goal. In LAMMPS the algorithm is contained within the goals of a much larger scope, requiring additional complexity so that code may be shared across algorithms, etc.

As stated above, this computes the forces acting upon each atom, and therefore the actions of each atom (potentially) on the others. This coupling of actions is reflected in a



(a) Neighbors



(b) Proportional difference

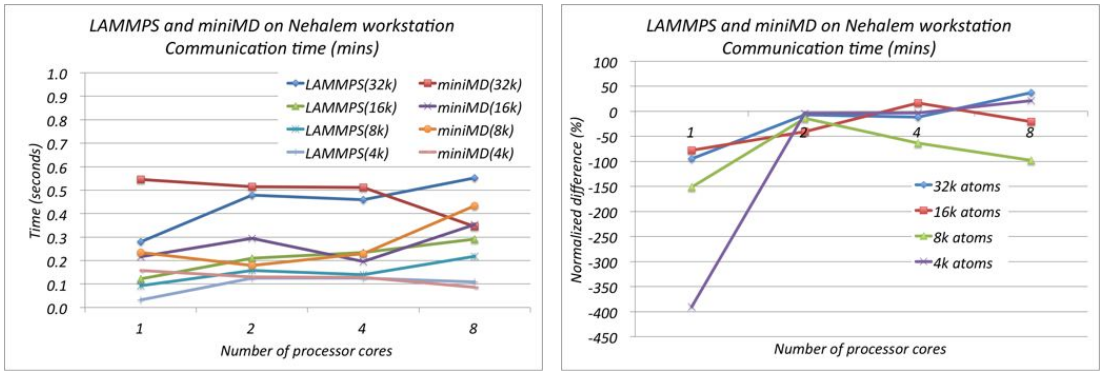
Figure 5.6. Performance, LAMMPS and miniMD : Neighbor time, strong scaling

coupling of the computation. For example, whereas `f[i][·]` and `f[j][·]`...This has implications with regard to the computational capabilities of current processors. Vectorization mechanisms, such as the Streaming SIMD Extensions (SSE) and the Advanced Vector Extensions (AVX) instruction sets can effectively manage these sorts of interactions. Conversely, threading approaches, such as pthreads [8] and OpenMP [9], is problematic due to potential runtime coupling of `f[{i,j}][·]`.

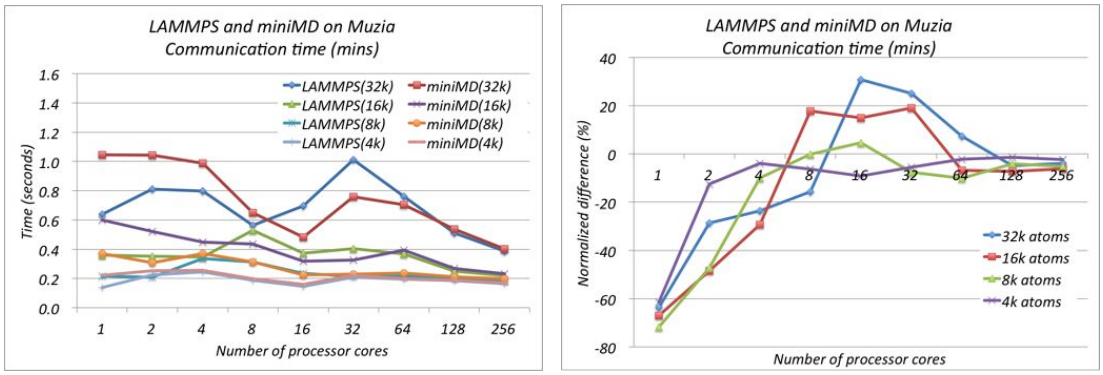
5.2 A Semiconductor Device Simulation code

Charon is a semiconductor device simulation computer program[2] [15, 23] developed at Sandia National Laboratories. It is a transport reaction code used to simulate the performance of semiconductor devices under irradiation. Although it employs both a finite element method

[2] `http://charleston.sandia.gov/Charon/`



(a) Inter-process Communication



(b) Proportional difference

Figure 5.7. Performance, LAMMPS and miniMD : Communication time, strong scaling

(FEM) and finite volume method (FVM) discretization, this work will focus on the FEM discretization. The coupled system of nonlinear partial differential equations (PDEs) describing the drift-diffusion model is shown in Equation 5.3 in residual form.

$$
\begin{aligned}
R_\psi &= \lambda^2 \bigtriangledown \cdot(\epsilon_r \mathbf{E}) - (p - n + C) = 0, \\
R_n &= \frac{\delta n}{\delta t} + \bigtriangledown \cdot (\mu_n n \bigtriangledown \psi) - \bigtriangledown \cdot (D_n \bigtriangledown n) + G = 0, \\
R_p &= \frac{\delta p}{\delta t} - \bigtriangledown \cdot (\mu_p p \bigtriangledown \psi) - \bigtriangledown \cdot (D_p \bigtriangledown p) + G = 0.
\end{aligned}
\tag{5.3}
$$

A stabilized FEM discretization strategy is designed to control the instability of the

Number of processor cores	L(32k)	MD(32k)	L(16k)	MD(16k)	L(8k)	MD(8k)	L(4k)	MD(4k)
	32k atoms		16k atoms		8k atoms		4k atoms	
Nehalem workstation								
1	0.4	0.3	0.3	0.00	0.5	0.1	0.2	0.5
2	0.5	0.6	1.1	0.00	1.2	0.8	0.4	0.9
4	1.0	2.9	0.6	0.01	1.7	0.8	0.7	1.0
8	4.3	1.0	1.0	0.11	7.1	10.2	4.8	0.0
Muzia								
1	0.84	0.14	0.83	0.06	0.03	0.01	0.01	0.03
2	0.67	0.13	0.03	0.05	0.07	0.09	4.28	0.05
4	0.05	0.13	0.03	0.14	5.26	0.06	5.92	0.03
8	0.05	0.18	0.11	0.17	1.80	0.09	2.72	0.06
16	0.07	0.04	1.72	0.02	6.54	0.03	0.18	0.04
32	0.42	0.98	1.45	0.18	0.38	0.15	0.67	0.11
64	4.02	1.69	1.97	1.51	0.21	1.58	0.31	0.12
128	0.98	0.90	1.73	0.53	0.04	0.62	1.70	0.27
256	0.97	0.21	0.84	0.65	0.83	0.25	0.03	0.12

Table 5.1. Standard deviations, as a percentage of the time, for LAMMPS and miniMD experiments

Galerkin formulation, resulting in the weak form shown in Equation 5.4.

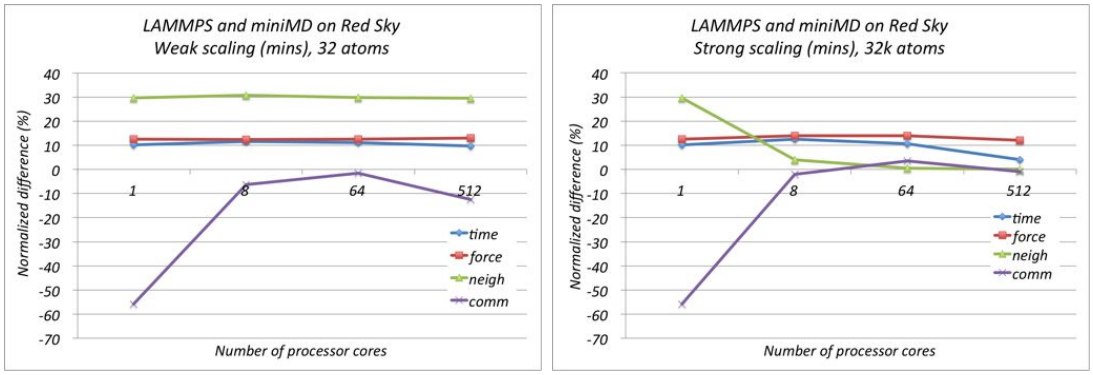
$$\begin{aligned} F_\psi &= \int R_\psi \phi d\Omega = 0, \\ F_n &= \int_\theta R_n \phi d\Omega - \sum_e \int_{\Omega_e} \tau_n [\mu_n \mathbf{E} \cdot \bigtriangledown_\phi] R_n d\Omega = 0, \\ F_p &= \int_\theta R_p \phi d\Omega + \sum_e \int_{\Omega_e} \tau_p [\mu_p \mathbf{E} \cdot \bigtriangledown_\phi] R_p d\Omega = 0. \end{aligned} \tag{5.4}$$

Finite element discretization of these equations in space on an unstructured mesh produces a sparse, strongly coupled nonlinear system. These equations are solved using a Newton-Krylov approach, resulting in a large sparse linear systems of the form

$$AM^{-1}(Mx) = b, \tag{5.5}$$

for $A \in \mathbb{C}^{N \times N}$, x and $b \in \mathbb{C}^N$, and some preconditioner M.

The linear systems are solved either using BiCGSTAB [41] (typically when the system is reasonably well-conditioned) or GMRES [34], without restart, when the system is poorly conditioned. A multigrid preconditioner [13], with local incomplete factorization as smoothers significantly improves scaling and performance [22].



(a) Weak scaling

(b) Strong scaling

Figure 5.8. Performance: LAMMPS and miniMD, 32k atoms, on Red Sky.

The code, written using C++, is configured for parallel computation using the MPI-everywhere model. The Trilinos [17] solvers are employed, with the Trilinos Aztec library [39, 16] providing the Krylov solvers, and the Trilinos ML library [13] providing the multigrid preconditioner.

An example 2D steady-state drift-diffusion solution is illustrated in Figure 5.9 for a bipolar junction transistor (BJT). The base, emitter and collector are located in the upper left corner, upper right corner and bottom edge respectively. The colors represent the electric potential; red denotes high voltage and blue denotes low voltage.

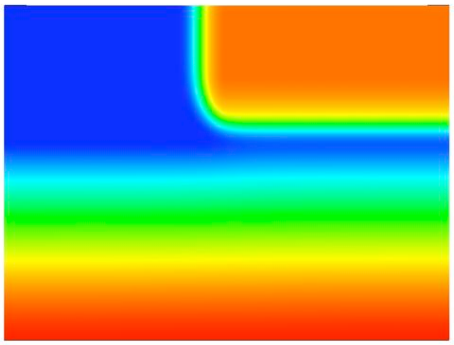


Figure 5.9. Charon steady-state solution

MiniFE is intended to mimic the finite element assembly and linear solution for a problem on unstructured meshes. MiniFE computes the element diffusion matrix for the steady state

conduction equation by solving

$$(K_{12}^e)_{xy} = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} k_{xy}\left(J_{11}^*\frac{\partial\psi_1}{\partial\xi} + J_{12}^*\frac{\partial\psi_1}{\partial\nu} + J_{13}^*\frac{\partial\psi_1}{\partial\zeta}\right)$$

$$\cdot\left(J_{21}^*\frac{\partial\psi_2}{\partial\xi} + J_{22}^*\frac{\partial\psi_2}{\partial\nu} + J_{23}^*\frac{\partial\psi_1}{\partial\zeta}\right)|J|d\xi d\nu d\zeta$$

and

$$I = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} F(\xi,\nu,\zeta)d\xi d\nu d\zeta$$

where

$$I \approx \Sigma_{I=1}^{M}\Sigma_{J=1}^{N}\Sigma_{K=1}^{P}F(\xi_I,\nu_J,\zeta_K)W_I,W_J,W_K,$$

as described in [33].

It assembles finite-element matrices into a global matrix and vector, then solves the linear-system using the Conjugate Gradient method [20]. Each finite-element is a hexahedron with 8 vertex-nodes. These equations are solved on a three-dimensional box of hexahedra. The

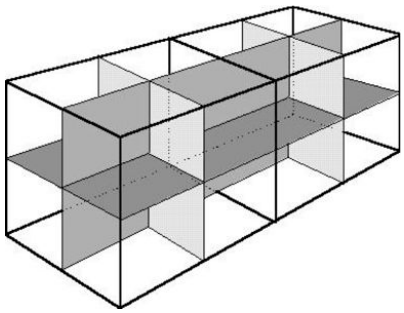


Figure 5.10. miniFE domain

Three-dimensional domain of hexahedra (This is a placeholder image! http://www.andrew.cmu.edu/user/sowen/survey/stc.jpg

domain dimensions are in terms of elements. For example, a $2 \times 2 \times 2$ box describes eight elements (illustrated in Figure 5.11), each of which has eight nodes, so it is a $3 \times 3 \times 3$ node domain (27 nodes). The coordinate origin is at the corner of the global box where $x = 0, y = 0, z = 0$. The box extends along the positive x-axis, positive y-axis, and the *negative* z-axis. Each node corresponds to a row in the matrix. A global identifier, assigned using coordinates and global box dimensions, adds coding convenience to some aspects of matrix-structure generation and finite-element assembly.

The domain is partitioned using the Recursive Coordinate Bisection method [6], and thus some processors own non-contiguous blocks of global node identifiers. Since it is convenient for matrices and vectors to store contiguously- numbered blocks of rows, global node

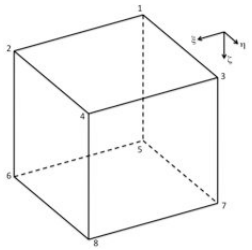


Figure 5.11. miniFE hexahedral finite element

identifiers are mapped to a separate space of row numbers such that each processor's nodes correspond to a contiguous block of row numbers.

The code, written using C++, is configured for parallel computation using the MPI-everywhere model. MiniFE has also been configured for computation on multicore nodes, including pthreads and Intel Threading Building Blocks (TBB) for homogeneous multicore and CUDA for GPUs.

5.2.1 Model abstractions

MiniFE solves a scalar equation, and therefore there is one degree of freedom per mesh node. Charon steady-state drift-diffusion problems have three degrees of freedom (DOF) per mesh node.

Runtime for typical Charon problems is focused in the Newton-Krylov solver. CG in MiniFE is intended to be sufficiently realistic to be representative of the performance of the Krylov solver portion in an application code, e.g. to be representative of the scaling of a single Krylov iteration. As miniFE solves a completely different set of physics, it is not intended to predict the either the number of Krylov iterations required or the number of Newton steps required. For our studies, the number of Krylov iterations for miniFE and a Newton step of Charon will be the same. Answers provided from miniFE and Charon will be completely different, however the goal is to see what insight into Charon performance miniFE can provide.

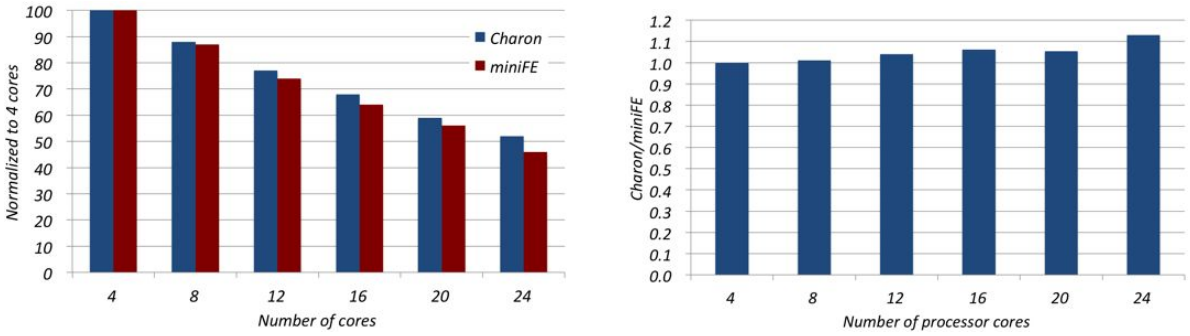
5.2.2 Performance Domain

D_1 :Node memory bandwidth
D_2 :Cache hit-to-miss ratio
D_3 :Weak scaling

We examine performance on three distinct architectures. Cielo, a Cray XE6, consists of dual-socket 8-core AMD Opteron Magny-Cours processor based nodes connected by a custom Gemini network configured as a three dimensional torus. Chama consists of dual-socket 8-core Intel Sandy Bridge processor based nodes connected by a Qlogic InfiniBand network configured as a fat tree. Red Sky consists of dual-socket quadcore Intel Nehalem processor based nodes connected by a Mellanox InfiniBand network configured as a three dimensional torus. We supplement these machines with related architectures that allow for more flexible experiments.

5.2.3 Diagnostic: Node memory bandwidth

Memory bandwidth within a multicore processor based node is seen as having a significant impact on the performance of an application. A typical means for exploring this issue is to vary the number of processor cores employed on the node and comparing the resulting performance efficiency. Figure 5.12(a) illustrates the results of this experiment applied to the solver phases on a Cray XE6 node configured using dual-socket 12-core AMD Opteron Magny-Cours processors. As has been observed in a variety of cases [11, 22, 1], the efficiency

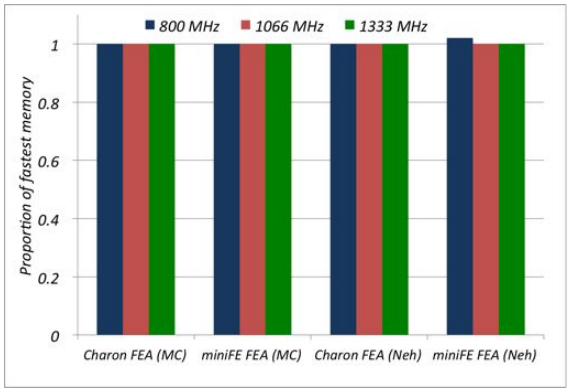


(a) Efficiency (b) Metric

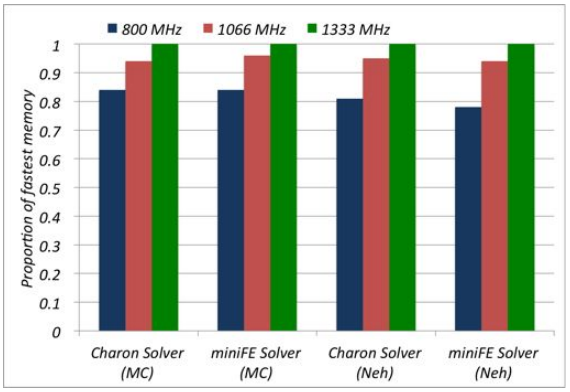
Figure 5.12. Effects of the number of cores per node on the FEA and solver phases of Charon and miniFE.

of each processor decreases as the number of cores per node increases. A proportional comparison (Figure 5.12(b)) reveals that the responses by Charon and miniFE are within about 13% at worst, suggesting that the miniapp is predictive of the effects of memory bandwidth on Charon. However, in a validation study, stronger evidence is needed to make this claim.

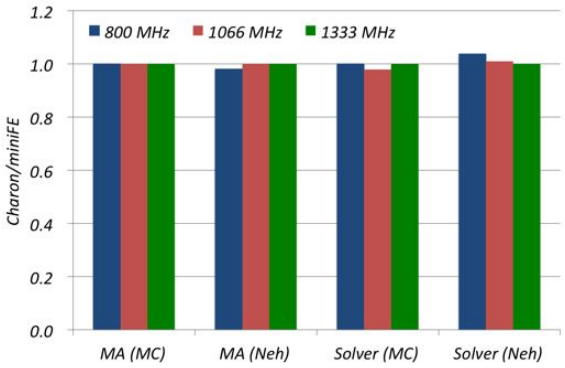
Using a dual-socket quadcore Intel Nehalem 5560 clocked at 2.8 GHz processors and a dual-socket 8-core AMD Magny-Cours 6136 clocked at 2.4 GHz, experiments were configured to better focus on memory bandwidth. The machines were configured to provide memory speeds of 800 MHz, 1066 MHz, and 1333 MHz. Results, illustrated in Figure 5.13, show



(a) FEA, normalized data



(b) Solver, normalized data



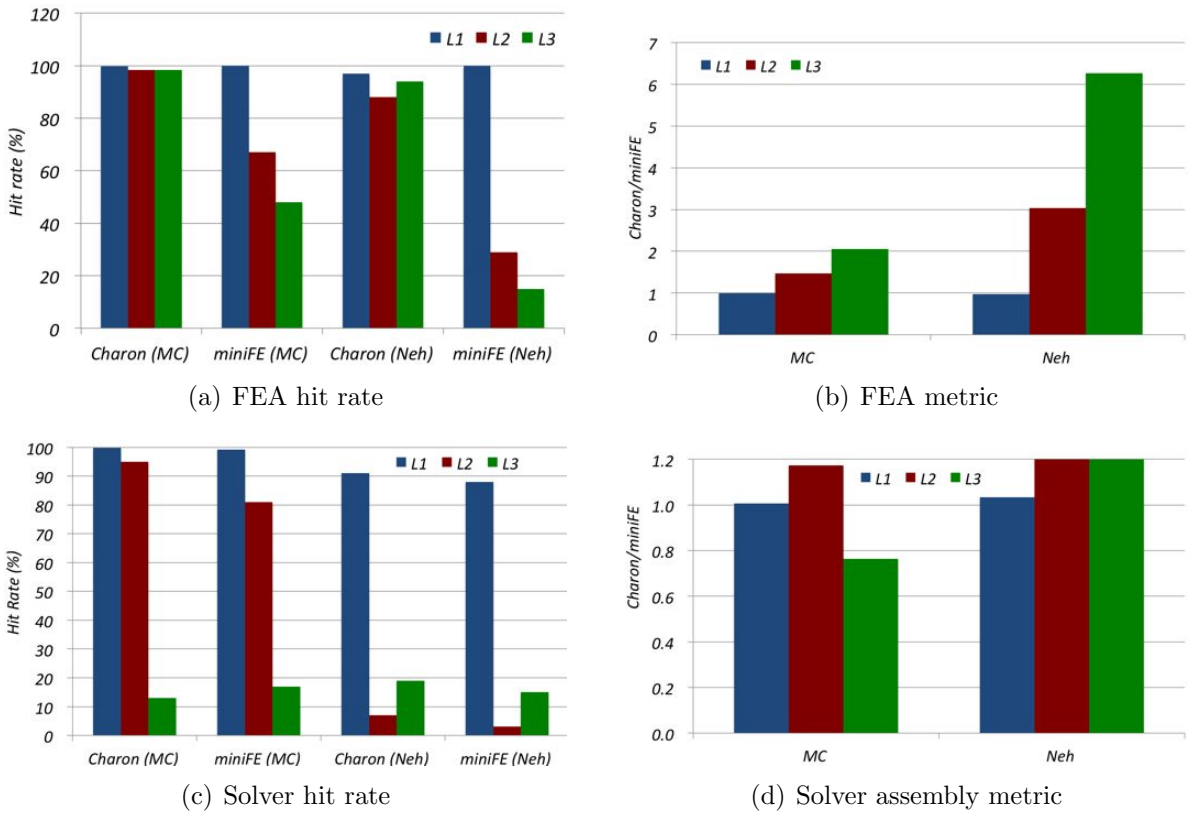
(c) Metric

Figure 5.13. Effects of memory speeds on the FEA and solver phases of Charon and miniFE. Performance is relative to 1333 MHz.

that the FEA phases for miniFE and Charon are not impacted by the change in bandwidth, while their solvers are. A proportional comparison (Figure 5.13(c)) shows miniFE is within 4% of all measures of Charon, leading us to claim that miniFE is predictive of Charon with regard to on-node memory bandwidth.

5.2.4 Diagnostic: Cache performance

The next diagnostic considers cache performance, again with the separation of between the FEA and solver phases, and again using the Nehalem and Magny-Cours nodes, each with three levels of cache (L3 is shared across cores in a socket). The hit rate, defined as the proportion of the number of times the processor finds needed data in a cache with the total number of times it looks for data in that cache, plays a significant role in processor performance. Results are shown in Figure 5.14. For the FEA phase, Charon and miniFE



(a) FEA hit rate

(b) FEA metric

(c) Solver hit rate

(d) Solver assembly metric

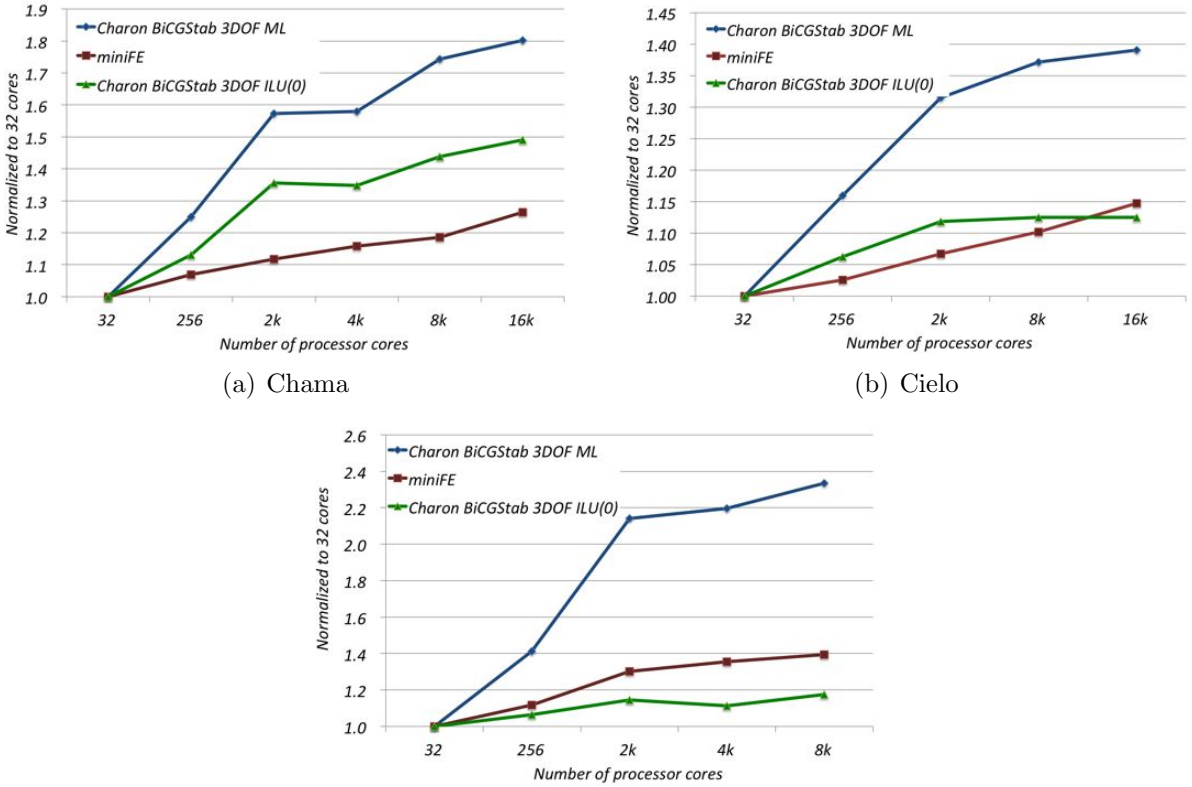
Figure 5.14. Cache behavior of the FEA and solver phases of Charon and miniFE.

show strong use of level 1 cache, with a proportional difference of no more than 3%. However, level 2 and 3 hit rates are significantly different, with miniFE 3 and 6 times, respectively, from Charon, leading us to claim that the cache performance of FEA in miniFE is not predictive of that for Charon. For the solver phase, we believe that cache performance is predictive. Although the thresholds for acceptance for level 2 and 3 are arguably high (20%) the trends are clear.

Most interesting is that the Charon/Aztec solver's surprisingly low level 2 hit rate seen on the Nehalem is also seen with miniFE. Given that this is unexpected given Magny-Cours and other past observations, care must be taken with regard to attribution. It is possible that measurement intrusion is to blame, or perhaps a hardware configuration issue. Additional experiments are required to make strong causal claims, since it is possible that measurement intrusion is to blame, or perhaps a hardware configuration issue.

5.2.5 Diagnostic: Weak scaling

Next we examine weak scaling characteristics of Charon and miniFE up to 16k core counts. Diagnostics include the Charon/Aztec BiCGSTAB solver with two preconditioning strategies, an incomplete factorization algorithm with no fill (ILU(0)) and a multilevel (ML) algorithm. Results for each are analyzed in comparison to miniFE, which does not employ a preconditioner. The general idea is that Krylov solvers perform common computations (e.g. addition and scaling of vectors, inner products, and sparse matrix-vector products. Further, applications typically use a breadth of preconditioners, so our goal is to understand where specificity is required and where it is not necessary. Performance is illustrated in Figure 5.15. We have not yet determined an effective means for analytically comparing scaling be-



(a) Chama

(b) Cielo

(c) Red Sky

Figure 5.15. Relative scaling of solvers

havior. Instead we can reason about the curves by first noting that performance is different on different architectures, which we speculate is a function of the difference preconditioning strategies. Although the difference between miniFE and Charon with ML preconditioning is large, this is not reason enough to reject the relationship. Instead, we claim that miniFE is *not* predictive of Charon with ML because miniFE does not include the sorts of compu-

tations found in ML. Further, an analysis of the interprocess message passing requirements shows that ML sends over 40% more message per core than do the other configurations.

The difference between Charon with ILU(0) preconditioning and miniFE is less clear, with reasoning driven from the position in the codesign space. For example, from the perspective of some hardware architects, these two approaches are not predictive. However, from the perspective of an algorithm developer perhaps investigating new programming models, miniFE performance could be reasonably predictive. Even with the Charon/Aztec BiCGSTAB solver being similar to CG in miniFE, there still are substantial differences between the two cases, such as the signficant difference in how the domain decomposition is performed. Clearly further investigation into this issue is needed. Therefore we assign this diagnostic a *caution* assessment.

5.2.6 Discussion

One of the intentions of miniFE is to be representative of the performance of a Krylov solver employed in an implicit finite element application code. For the case where miniFE models a completely different set of physics than an application code such as Charon, it is not intended to predict either the number of Krylov iterations required or the number of Newton steps required for the application code. From the memory bandwidth studies (both changing the channel frequency and increasing contention by increasing the number of processes) and cache hit rate studies, miniFE seems to do a good job being representative of the performance of the Charon/Aztec Krylov solver within a single compute node. But the scaling studies for large numbers of compute nodes demonstrate the differences between miniFE and Charon. As miniFE does not have a preconditioner, it clearly was not intended to model an application code that employs a multigrid preconditioner. This is a concern for future large-scale simulations that will require a multilevel/multigrid preconditioner in order to achieve convergence. Regarding the disparity between miniFE and Charon/Aztec without a multilevel preconditioner, further investigations into the impact of differences such as the difference in domain decomposition is necessary.

5.3 A Shock Physics code

CTH is a multi-material, large deformation, strong shock wave, solid mechanics code developed at Sandia National Laboratories [19]. CTH has models for multi-phase, elastic viscoplastic, porous and explosive materials, using second-order accurate numerical methods to reduce dispersion and dissipation and produce accurate, efficient results.

Two distinct problems are commonly modeled by CTH. The meso-scale impact in a confined space problem is computationally well-balanced across the parallel processes. This problem involves 11 materials, inducing the boundary exchange of 75 variables. The shaped charge problem, illustrated in Figure 5.3, involves four materials, inducing the boundary

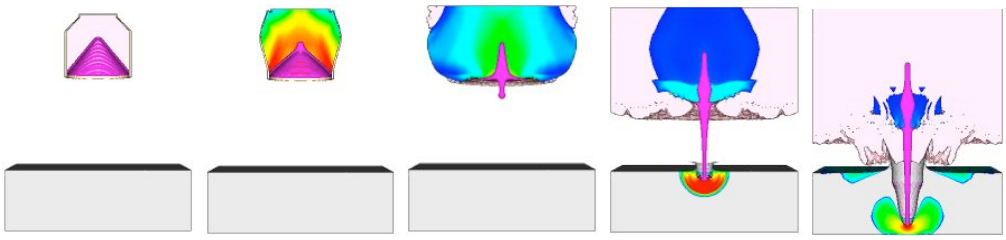


Figure 5.16. CTH shaped charge simulation

Time progresses left to right.

exchange of 40 variables. (This problem was used in the acceptance testing for the NNSA ASC campaign's latest capability computer, Cielo [10].)

Several times each time step boundary information is aggregated and exchanged with up to six neighbors in the grid of processors. Also, several reductions (mostly `MPI_Allreduce` on 8-byte data) during each time step.

MiniGhost is serving as a proxy for CTH, in terms of the following:

A broad range of physical phenomena in science and engineering can be described mathematically using partial differential equations. Determining the solution of these equations on computers is commonly accomplished by mapping the continuous equation to a discrete representation. One such solution technique is the finite differencing method, which lets us solve the equation using a difference stencil, updating the grid as a function of each point and its neighbors, presuming some discrete time step. The algorithmic structure of the finite difference method maps naturally to the parallel processing architecture and single-program multiple-data (SPMD) programming model. For example, on a regular, structured grid, $O(n^2)$ computation is performed, with nearest neighbor $O(n)$ inter-process communication requirements.

On parallel processing architectures, these sorts of computations require data from neighboring processes. Inter-process communication is typically abstracted into some sort of functionality that may be loosely described as *boundary exchange* (likewise also called ghost- or halo-exchange) This notion of mapping a continuous problem to discrete space and the inter-process communication requirement induced by spacially decomposing the grid across parallel processes adheres to the bulk-synchronous parallel programming model (BSP [40]), arguably the dominant model for implementing high performance portable parallel processing scientific applications [4].

In the BSP/message aggregation (BSPMA) model, data from multiple (logical) memory locations are combined into a user-managed array with other data, then subsequently transmitted to the target process. This step incurs three costs: memory utilization (the message buffers), on-node bandwidth (copies into the buffer), and synchronization (leading up to and

including the data transfer).

Several times within a time step boundary information is aggregated and exchanged with up to six neighbors in the grid of processors. For the shaped-charge problem these messages average 4.1 MB and for the meso-scale problem these messages average 10.4 MB. Process 0 and a couple of other processors near it for the shaped charge problem have more work since they are the genesis of the explosion and thus have additional work relative to the other processes. The runtime trace shows significantly less waiting time than the other cores.

We compared performance of miniGhost and CTH on a Cray XT5. Results are shown in Figure 5.17. Run in weak scaling mode on up to 1,024 processor cores (this XT5 is a

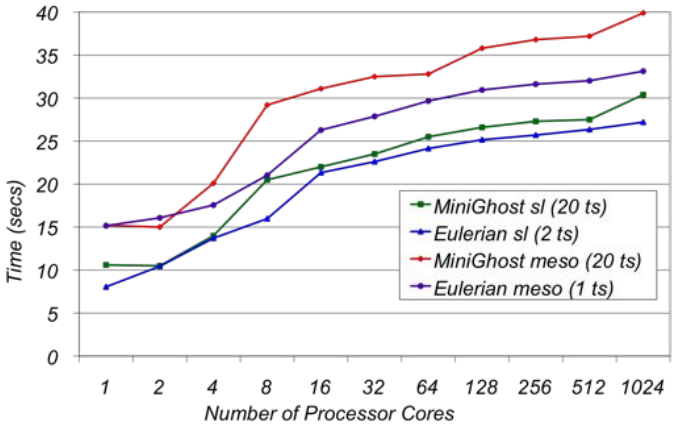


Figure 5.17. Performance: CTH and miniGhost

dual-socket AMD Opteron Istanbul hex-core node based machine with SeaStar interconnect, details in [42]), miniGhost tracks CTH performance reasonably well.

Although the above does not fully validates miniGhost as representing CTH performance in the defined context, combined they offer substantial encouragement that the connection is worthwhile. In particular, though, miniGhost does provide a contextual means for exploring various boundary exchange configurations such as those listed in the next section.

5.3.1 Model abstractions

As stated throughout this paper, miniapps are not intended to capture all aspects of a particular application. Instead, they are designed to represent issues that critically impact the runtime characteristics of large scale application programs. In addition to enabling a stronger focus on a limited set of important issues in a particular application, this approach allows a miniapp to be representative of more than one application, and in some cases, represent a class of applications and algorithms. Toward that end, below is a list of some

key distinctions between miniGhost and CTH.

- **stencils** CTH is a finite volume code, and thus it has values that are based at cells and at the nodes. CTH does several things with the values of variables in both the cell being computed and the surrounding cells. Portions of the code simply use the values of different variables in a given cell to calculate new values of those and other variables in that cell (for example equation of state calculations). Other portions use the value of variables in a cell and those neighboring cells in some direction to calculate new values (such as advection). And other portions of the code use values from variables in all 26 neighboring cells to calculate values of variables in a cell (such as interface tracking). This can alter cache and other behavior. MiniGhost computations in some sense, though, provide a meaningful context for the interprocess communication, which looks similar to a finite difference code.

- **ack to ensure receive posted** In CTH, prior to sending a message containing boundary data to a neighbor, an MPI process waits for a message from the target processes, which alerts the sending process that the matching receive is posted. The intent is to avoid unexpected messages, potentially a serious issue given the typically large amount of data transmitted. This is a meaningful approach on some architectures, such as Red Storm [37], but is of no help on most, which include a built-in acknowledgement handshake prior to sending messages. Further, the MPI specification provides functionality managing this. Regardless, our intent is to test the capabilities of the MPI implementation on the target architecture external of some means for adapting to the specific capabilities of a particular implementation. That said, one use of a miniapp is to provide a lower impact means for testing other approaches and ideas.

- **reductions** CTH makes several calls to MPI colletives (e.g. 90 for some problem sets, and typically to `MPI_Allreduce` with a small count input) throughout a time step. MiniGhost is configured to include this as an option, with the intent of injecting collective synchronization.

- **data structure** CTH manages material mesh data as sets of two dimensional slices, contained in a single pool of allocated memory. This memory management scheme is a relic of past language constraints. MiniGhost allocates distinct three dimensional arrays, each representing a material.

- **load imbalance and AMR** CTH is a multi-material code, so over time materials enter and exit cells, altering the computational load as well as interprocess communication requirements. Further, CTH also provides the option for adaptive mesh refinement (AMR), which focuses attention on cells under some specified condition. The cell is (evenly) divided into four new cells. If necessary, neighboring cells will be divided in order to maintain at most two neighboring cells. Future plans call for the incorporation load imbalance into miniGhost as a means of studying the effects of this behavior.

5.3.2 Performance Domain

MiniGhost is designed to capture the inter-process communication requirements of CTH. Therefore the diagnostics are defined to measure its message passing characteristics.

D_1 :Number of communication partners (neighbors)
D_2 :Number of messages per boundary exchange.
D_3 :Message volume (bytes per neighbor)
D_4 :Weak scaling

Two problem sets described above (shaped charge and meso-scale) provide the drivers for these measurements.

In order to ensure that miniGhost accurately reflects the inter-process communication behavior, it is important to understand the inter-process communication infrastructure, including physical interconnect, logical to physical process maps, system software, and ultimately to how the application manages its communication requirements.

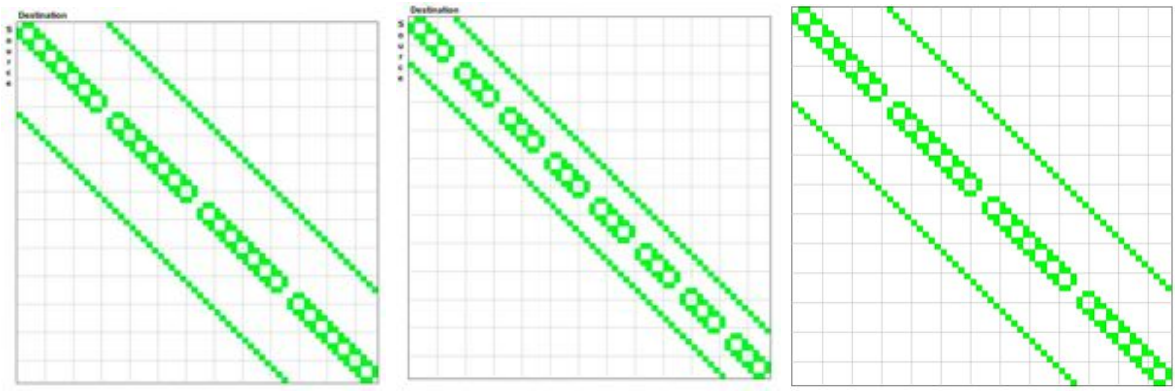
For example, when MPI communication is initiated, buffers are configured for managing message queues, unexpected messages, etc. In the distinct communication/computation phases of the full application, the compute stage touches enough data to ensure that all communication data structures have been flushed from the processor cache hierarchy and must be refetched from main memory upon the initiation of the communication phase. The work in miniGhost must be enough in order for this to occur. Thus for example, a single variable weak scaling problem of dimension $100 \times 100 \times 100$ in 8-byte precision means that an eight MByte variable is operated on, stored into another eight MByte variable. Thus 16 MBytes of memory has been traversed by the processor, effectively flushing a cache of that size.

Alternative methods for moving the data are being explored using miniGhost, and therefore the experimenter must take this into consideration.

5.3.3 Diagnostics: Boundary exchange characteristics

MiniGhost can be configured to match the number of communication partners, the number of variables, and the dimensions of those variables such that the boundary exchange is the same as with CTH. Figure 5.18 illustrates the communication patterns of two important CTH problems with that of miniGhost.

Diagnostics D_1, D_2, and D_3 were configured, and were verified, to be equivalent between miniGhost and the CTH problems under consideration.



(a) CTH shaped charge (b) CTH mesoscale (c) miniGhost

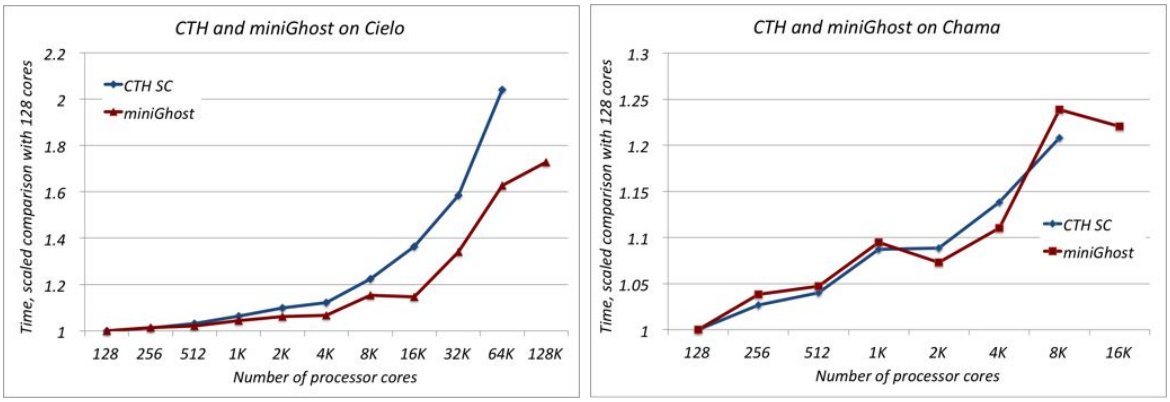
Figure 5.18. Boundary exchange communication patterns for the CTH shaped charge problem and miniGhost. The process in row i sends data to the process in column j.

5.3.4 Diagnostic: Weak Scaling

CTH provides a typical example of a code team adapting to computing architectures: in order to avoid message latencies and exploit global bandwidth, computation is performed across as many variables as possible before an boundary exchange across those variables can be consolidated in to a single message per neighbor. But in a recently completed broad-based study of Cielo capabilities [3], and reproduced on Chama, the nearest neighbor boundary exchange encountered significant scaling degradation beyond 8,000 processor cores. This issue is predicted by miniGhost, illustrated in Figure 5.19.

The problem was traced to the mapping of the parallel processes to the three dimensional torus topology, illustrated in Figure 5.20. Neighbors in the x direction required a maximum of one hop and in the y direction a maximum of two hops. But the number of hops across the network (referred to as the *Manhattan distance*) was shown to increase significantly in the z direction. This combined with the very large messages of a typical CTH problem set (e.g. for the "shaped charge" problem, 40 three dimensional state variable arrays generated message lengths of almost 5 MBytes) resulted in poor scaling beginning at 8k processes, a trend that accelerated after 16k processes.

In response, we implemented a means by which the parallel processes could be logically re-mapped to take advantage of the physical locality induced by the communication requirements. In the normal mode, CTH (and miniGhost) assigns blocks of the mesh to cores in a manner which ignores the connectivity of the cores in a node. On Cielo, as with other Cray X-series architectures, cores are numbered consecutively on a node, and this numbering continues on the next node. Blocks of the mesh are assigned to cores by traversing the blocks of the mesh in the x direction of the mesh starting at one corner of the mesh. Once those



(a) Cielo (b) Chama

Figure 5.19. Weak scaling of CTH and miniGhost on Cielo and Chama

blocks are assigned, the next block assigned is the block one over in the y direction of the mesh from the first block assigned. The mesh is again then traversed in the x direction and blocks are assigned to cores. This process is continued until there are no more blocks in the y direction. The next block assigned is then the first block in the z direction from the first block assigned. The blocks of the mesh with this z value are then assigned as the first blocks were assigned. This process is then repeated until all blocks in the mesh have been assigned to cores in the machine.

Our remapping algorithm assigns blocks of the mesh to the cores of the machine by groups. On Cielo, a group of blocks consists of a $2 \times 2 \times 4$ group of blocks. These blocks are then assigned to nodes as above. The result is a slight increase in the average hop counts in the x and y directions, but a significant decrease in the average hop count in the z direction. A comparison of the number of hops between the two approaches is shown in Table 5.2.

This remapping strategy results in a significant improvement in scaling performance, illustrated in in Figure 5.21(a). Figure **??** shows that this is attributable to controlling the time spent sending data in the z direction. We include the time spent in the reduction sum across each grid variable (inserted after computation on each variable to add application realism as well as a synchronization point), illustrating that this functionality is not the source of the issue, scaling well regardless of the processor mapping. We do see indications of the issue at the highest processor counts, though it is less pronounced. This remapping was incorporated into CTH, the results of which are described in Appendix **??**.

As discussed in the related work section above, we are exploring ways for incorporating these ideas into a more general interface. We are also exploring the use of `MPI_Datatype` in handling the non-contiguous (but patterned) face data.

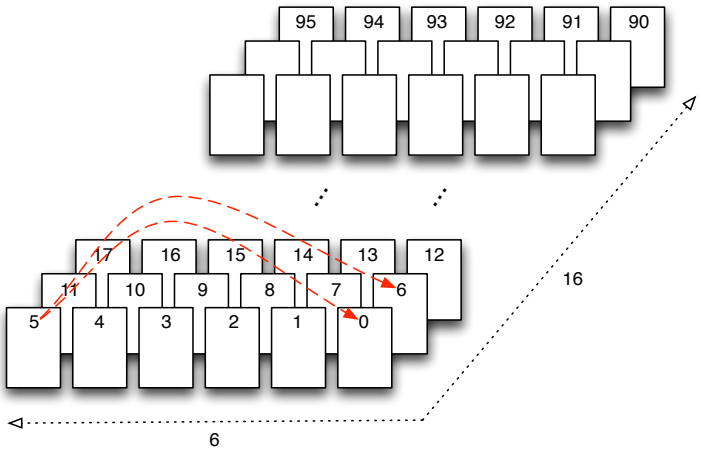


Figure 5.20. Cielo process map

5.3.5 Alternative communication strategies

Node interconnects are also evolving, driven by new node architectures as well as cost and energy conservation goals, encouraging exploration of new approaches within the context of application requirements. Interconnects are designed as a balance of global bandwidth (the ability of the interconnect to move data), inject bandwidth (the ability of the NIC to put data onto the interconnect), and injection rate (the ability of a node to place messages onto the NIC). Global bandwidth typically incurs the highest costs, both in terms of money and power consumption, and therefore we are preparing for a proportional decrease in that capability.

MiniGhost includes an application-relevant infrastructure for exploring alternative boundary exchange configurations [5]. The first configuration mimics that of CTH, which we call Bulk Synchronous Parallel with Message Aggregation (BSPMA), was used above, illustrated in Figure 5.22. The second, called Single Variable Aggregated Faces (SVAF) transmits data as soon as computation on a variable is completed, and thus six messages are transmitted for each variable (up to 40), one to each neighbor, each time step. (Looking at Figure 5.22, this eliminates the inner `END DO` and `DO I = 1, NUM_VARS`.) The two $x-y$ faces are contiguous in memory, so each may be directly sent using a call to a single MPI function. The other four faces are aggregated into buffers, resulting in four messages to their neighbors. A third mode, called single variable, contiguous pieces, computational overlapping mode (SVCP), is designed for use on architectures that are strongly biased toward significantly increased message injection rates and injection bandwidth, a trend we see developing but not yet to the extent of supporting this configuration using MPI [35].

BSPMA and SVAF have been configured for MPI-everywhere as well as MPI+OpenMP. For the latter on Cielo and Curie, its best configuration is four MPI ranks on each node,

Number of MPI ranks	Regular Order			Reordered		
	X	Y	Z	X	Y	Z
16	0.0	0.0	0.0	0.0	0.0	0.0
32	0.0	0.0	0.0	0.0	0.0	0.0
64	0.0	0.0	0.3	0.0	0.3	0.0
128	0.0	0.0	1.0	0.0	0.5	0.0
256	0.0	0.0	1.0	0.0	0.5	0.3
512	0.0	0.1	2.0	0.0	0.6	0.4
1024	0.0	0.3	2.1	0.2	1.0	0.7
2048	0.0	0.3	2.7	0.3	1.2	1.2
4096	0.0	0.3	3.7	0.3	1.2	1.2
8192	0.0	0.5	5.1	0.2	1.1	2.0
16384	0.0	0.5	4.9	0.2	1.1	2.2
32768	0.0	0.5	5.6	0.2	1.1	2.5
65536	0.0	1.1	10.2	0.2	1.6	2.8
131072	0.0	1.1	10.1	0.2	1.6	3.1

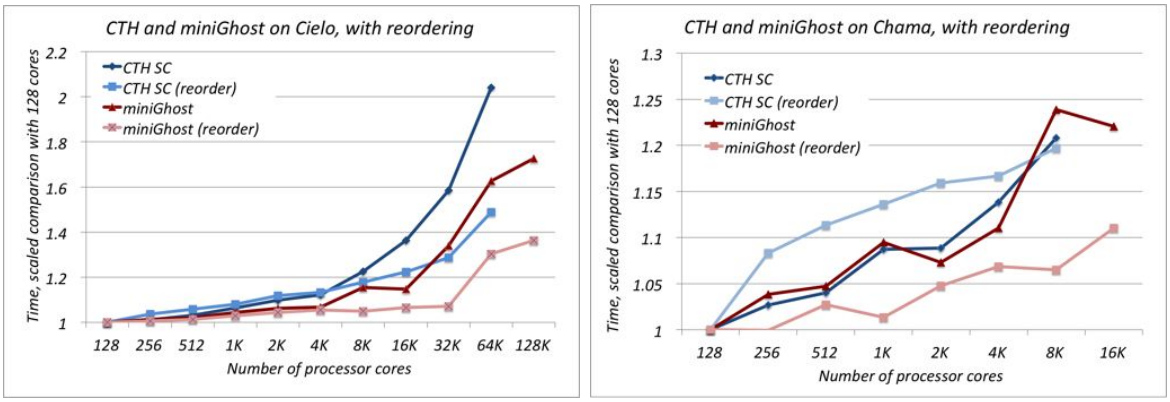
Table 5.2. miniGhost average hop counts on Cielo

each spawning four OpenMP threads. Note that this increases the size of each message in comparison with the MPI-everywhere version. Because of ghost cells, the message size in two directions almost doubles and the message size in the third direction almost quadruples. This is because the number of cells in two of the three directions is doubled. The size of the message is based the size of the face with ghost cells, so if the size of a face is $x \times y$ cells for the MPI everywhere case, the size of the message is $(x + 2) * (y + 2)$. If the number of cells in one of these directions (say y) is doubled, then the size of the message is $(x+2)*(2y+2)$, which is not quite double the original size, but close. Similarly, if the number of cells in both directions are doubled, then the size of the message is almost quadrupled.

Performance of these implementations on Cielo are shown in Figure 5.23. Effective mapping of processes to processors is again critical to achieving good scaling, and as the number of processors increases, SVAF becomes the best strategy. This is of significant interest since it reduces demand on costly global bandwidth by a factor of N, where N is the number of variables aggregated (40 for the shaped charge problem.)

5.3.6 Discussion

In some cases diagnostics are by themselves not very meaningful. However, they often provide meaning in the context of other diagnostics. For example, the number of communication partners (D_1) and messages per boundary exchange (D_2), when operating on static meshes, are expected to be (and are) the same between the app and miniapp. This then relates



(a) Cielo (b) Chama

Figure 5.21. Performance of MiniGhost with MPI-rank remapping on Cielo

to the interpretation of the message frequency and volume. The importance of D_1 and D_2 increases when the application is examined when executing on a dynamic mesh (i.e. Adaptive Mesh Refinement, AMR). Future work calls for determining the predictive capabilities of miniGhost as a static code with regard to CTH as an AMR code.

5.4 A Circuit Simulation code

Xyce is a circuit modeling tool [3] [21] developed at Sandia National Laboratories. It is designed to perform transistor-level simulations for extremely large circuits on large-scale parallel computing platforms of up to thousands of processors. Xyce is a traditional analog-style circuit simulation tool, similar to the Berkeley SPICE program[25].

Circuit simulation adheres to a general flow, as shown in Fig. 5.24. The circuit, described in a netlist file, is transformed via modified nodal analysis (MNA) into a set of nonlinear differential algebraic equations (DAEs)

$$\frac{dq(x(t))}{dt} + f(x(t)) = b(t), \tag{5.6}$$

where $x(t) \in \mathbb{R}^N$ is the vector of circuit unknowns, q and f are functions representing the dynamic and static circuit elements (respectively), and $b(t) \in \mathbb{R}^M$ is the input vector. For any analysis type, the initial starting point is this set of DAEs. The numerical approach employed to compute solutions to equation (5.6) is predicated by the analysis type.

[3] `http://xyce.sandia.gov/`

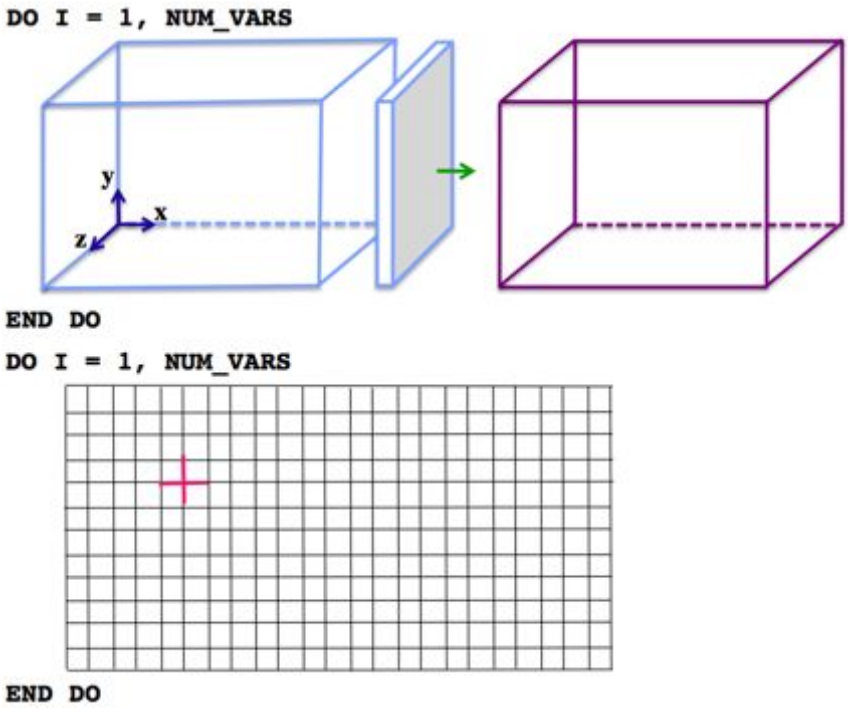
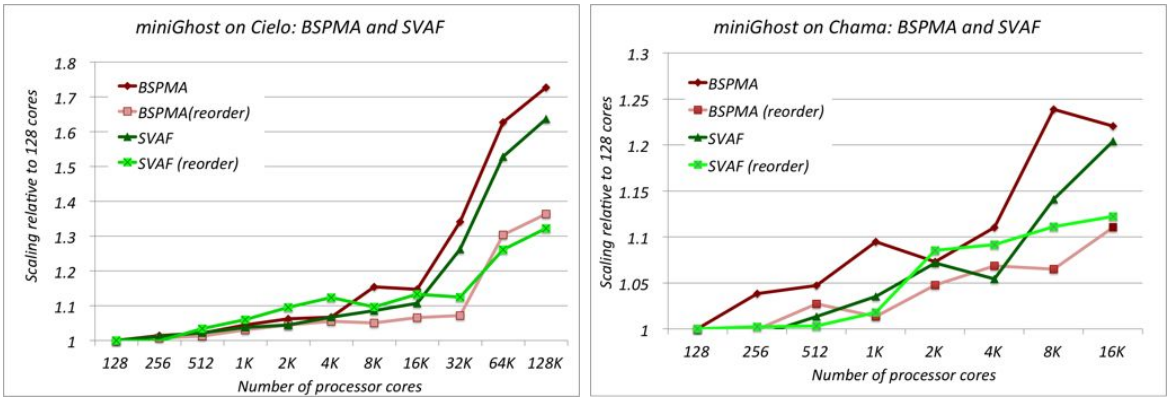


Figure 5.22. CTH boundary exchange and computation



(a) Cielo (b) Cielo

Figure 5.23. Performance of miniGhost Communication Strategies on Cielo

Circuit simulation can be coarsely divided into three phases that play a distinct role in the overall performance of a simulation. The first two phases result naturally from the evaluation

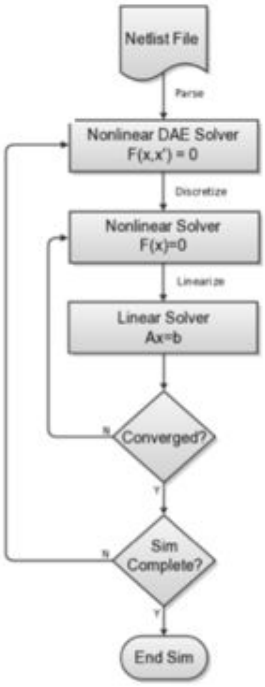


Figure 5.24. General Circuit Simulation Flow in Xyce

and the solution of the circuit equation (5.6) and are called the "Device Evaluation" and "Linear Solve" phases. In general, load balancing each of these two phases has competing objectives, indicated by Fig. 5.25, which requires a rebalancing of the problem between those phases.

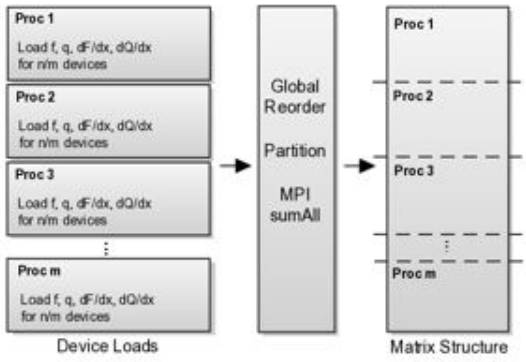


Figure 5.25. Different Load Balance/Partitioning for Device Evaluation and Linear Solve

The relative amount of time spent in each of those two phases is problem-dependent. For smaller problems, the device evaluation phase should dominate the runtime, especially when the circuit includes modern transistors. As the problem size increases, the linear solve phase will dominate, as it should scale super-linearly, while the device evaluations should scale linearly. This is because linear solution methods (whether they be direct or iterative) are generally communication intensive, while the communication volume required during the device evaluations is relatively small.

As a result, the device evaluation phase has historically been naively balanced by taking into account only the computational work required, while the matrix partitioning has been designed to minimize communication volume. How this communication volume is measured, and how it is optimized is an active area of research for many types of numerical simulation problems. Since the device evaluation and linear solve phases have different load balance requirements, Xyce has been designed to have completely different parallel partitioning for each. A simplified representation of this is shown in Fig. 5.25.

The third phase is "Parsing", where the hierarchical netlist file, describing the network elements and connectivity, is read in and the set of DAEs is constructed and partitioned across processors. In the total runtime of a simulation the netlist parsing doesn't take a large percentage, but the decisions made about partitioning devices over processors in this phase can possibly have a significant effect for emerging architectures, making it a phase worth studying.

Given the hierarchical structure possible in the netlist file, parsing is a largely serial

process, where devices are naively partitioned according to a "first-come-first-served" basis. This process is not guided by circuit topology or computational cost of the individual device evaluations, which can vary widely. This design has not been troublesome on the distributed-memory machines that Xyce was designed for. However, future architectures may prove this approach is too simplistic.

5.4.1 Model Abstractions

At this time, miniXyce is a simple linear circuit simulator with a basic parser that performs transient analysis on any circuit with resistors (R), inductors (L), capacitors (C), and voltage/current sources. The parser incorporated into this version of miniXyce is a single pass parser, where the netlist is expected to be flat (no hierarchy via subcircuits is allowed). Simulating the system of DAEs generates a nonsymmetric linear problem, which is solved using un-preconditioned GMRES [34]. The time integration method used in miniXyce is backward Euler with a constant time-step.

The development of the first version of miniXyce resulted in something closer to a compact application than a miniapp since more focus was put on the simulator returning the correct answer, than modeling performance characteristics of interest. Further analysis of Xyce has called out particular performance issues in the three phases discussed in Section 5.4. These issues will inspire enhancements to, and a second version of, miniXyce. For complex simulation codes, developing a representative miniapp may become an iterative process, where performance issues are investigated in order of an application-based priority.

5.4.2 Model Enhancements

Enhancements to the first version of miniXyce will focus on two of the three phases discussed in Section 5.4: "Parsing" and "Device Evaluation". The third phase, the "Linear Solve", shares performance characteristics and issues with other implicit application codes, like Charon. While this phase dominates the runtime for large-scale circuits, it is uncertain if focusing on that aspect of miniXyce will be a duplication of effort with miniFE. However, the "Parsing" and "Device Evaluation" phase are unique to circuit simulation. By focusing on these two phases, it is anticipated that miniXyce will provide a different performance characterization than any other miniapp. Alternatively, it will give Xyce a unique opportunity to explore the impact of naively partitioning the network and devices.

Chapter 6

Summary and future work

Miniapps provide a tractable means for rapid exploration of issues associated with effectively executing large scale scientific and engineering applications on current, emerging, and future architectures.

We presented a methodology for understanding the relationships between an application proxy (the miniapp) and the application in the manner in which it is intended to represent. Four distinct applications provided the means for demonstrating this link. Using a miniapp (miniMD) closely aligned to the application (LAMMPS) it is intended to represent, we could in some strong sense, calibrate this approach. MiniGhost provides a means for focusing on the inter-process communication requirements of an application (here CTH), with the computation serving to create a separation between the transmission of data between the parallel processes. Within certain limits, MiniFE provides a means to investigate linear solver performance for an implicit finite element method on unstructured mesh application code. Through this work it was determined that miniXyce requires additional analysis, design, and implementation in order for it to serve the performance goals of the Xyce team.

Another use of miniapps is to provide a tractable means for using simulators, such as the Structural Simulation Toolkit (SST). Because we don't expect simulator results to align perfectly with the miniapp experiments (just as they do no herein), this methodology would enable an analytic means for understanding how the simulator results map to the miniapp experiments. Here the baseline observations B would be the miniapp and the corresponding measurements A would be that predicted by the simulator. We expect to report on results of these sorts of experiments soon.

References

[1] S.R. Alam, R.F. Barrett, J.A. Kuehn, P.C. Roth, and J.S. Vetter. Characterization of Scientific Workloads on Systems with Multi-core Processors. In *IEEE International Symposium on Workload Characterization*, 2006.

[2] K. Alvin et al. On the Path to Exascale. *International Journal of Distributed Systems and Technologies*, 1(2):1–22, 2010.

[3] B.W. Barrett et al. Report of Experiments and Evidence for ASC L2 Milestone 4467 - Demonstration of a Legacy Application's Path to Exascale. Technical Report SAND2012-1750, Sandia National Laboratories, 2012.

[4] R.F. Barrett, S. Ahern, M.R. Fahey, R. Hartman-Baker, J.K. Horner, S.W. Poole, and R. Sankaran. A Taxonomy of MPI-Oriented Usage Models in Parallelized Scientific Codes. In *The International Conference on Software Engineering Research and Practice*, 2009.

[5] R.F. Barrett, C.T. Vaughan, and M.A. Heroux. MiniGhost: A Miniapp for Exploring Boundary Exchange Strategies Using Stencil Computations in Scientific Parallel Computing. Technical Report SAND2011-5294832, Sandia National Laboratories, May 2011. `https://software.sandia.gov/mantevo/publications.html`.

[6] M. J. Berger and S. H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. Comput.*, 36:570–580, May 1987.

[7] R. Brightwell, K.T. Pedretti, K.D. Underwood, and T. Hudson. SeaStar Interconnect: Balanced Bandwidth for Scalable Performance. *IEEE Micro*, 26:41–57, 2006.

[8] D. Buttlar, B. Nichols, and J.P. Farrell. *pthreads Programming*. O'Reilly & Associates, Inc., 1996.

[9] L. Dagum and R. Menon. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Computational Science and Engineering*, 5(1):46 –55, 1998.

[10] D.W. Doerfler, M. Rajan, C. Nuss, C. Wright, and T. Spelce. Application-Driven Acceptance of Cielo, an XE6 Petascale Capability Platform. In *Proc. 53rd Cray User Group Meeting*, 2011.

[11] J. Dongarra, D. Gannon, G. Fox, and K. Kennedy. The Impact of Multicore on Computational Science Software. *CTWatchQuarterly*, 3(1), February 2007.

[12] J. Dongarra and P. Luszczek. *Beautiful Code: Leading Programmers Explain How They Think*, chapter How Elegant Code Evolves with Hardware: The Case of Gaussian Elimination. O'Reilly, 2007.

[13] M.W. Gee, C.M. Siefert, J.J. Hu, R.S. Tuminaro, and M.G. Sala. ML 5.0 Smoothed Aggregation User's Guide. Technical Report SAND2006-2649, Sandia National Laboratories, 2006.

[14] A. Geist and S. Dosanjh. IESP Exascale Challenge: Co-Design of Architectures and Algorithms. *Int. J. High Perform. Comput. Appl.*, 23:401–402, November 2009.

[15] G. L. Hennigan, R. J. Hoekstra, J. P. Castro, D. A. Fixel, and J. N. Shadid. Simulation of Neutron Radiation Damage in Silicon Semiconductor Devices. Technical Report SAND2007-7157, Sandia National Laboratories, 2007.

[16] M. Heroux. AztecOO user guide. Technical Report SAND2007-3796, Sandia National Laboratories, 2007.

[17] M. A. Heroux et al. An Overview of the Trilinos Project. *ACM Transactions on Mathematical Software*, 31:397–423, September 2005.

[18] M.A. Heroux et al. Improving Performance via Mini-applications. Technical Report SAND2009-5574, Sandia National Laboratories, September 2009. `https://software.sandia.gov/mantevo/`.

[19] E.S. Hertel and others. CTH: A Software Family for Multi-Dimensional Shock Physics Analysis. In *Proceedings, 19th International Symposium on Shock Waves*, 1993.

[20] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Stand.*, 49:409–436, 1952.

[21] E.R. Keiter and others. Parallel Transistor-Level Circuit Simulation. In Peng Li, Luis Miguel Silveira, and Peter Feldmann, editors, *Advanced Simulation and Verification of Electronic and Biological Systems*. Springer, 2011.

[22] P.T. Lin and J.N. Shadid. Towards Large-Scale Multi-Socket, Multicore Parallel Simulations: Performance of an MPI-only Semiconductor Device Simulator. *Journal of Computational Physics*, 229(19):6804–6818, 2010.

[23] P.T. Lin, J.N. Shadid, M. Sala, R.S. Tuminaro, G.L. Hennigan, and R.J. Hoekstra. Performance of a Parallel Algebraic Multilevel Preconditioner for Stabilized Finite Element Semiconductor Device Modeling. *Journal of Computational Physics*, 228(17):6250–6267, 2009.

[24] N. Metropolis. The Beginning of the Monte Carlo Method. Technical report, Los Alamos National Laboratory, 1987.

[25] L. W. Nagel. SPICE 2, a Computer Program to Simulate Semiconductor Circuits. Technical Report Memorandum ERL-M250, University of California, Berkley, 1975.

[26] W. L. Oberkampf and C. J. Roy. *Verification and Validation in Scientific Computing*. Cambridge University Press, 2010.

[27] W. L. Oberkampf and T. G. Trucano. Verification and Validation in Computational Fluid Dynamics. *Progress in Aerospace Sciences*, 38, 2002.

[28] M. Pilch, T.G. Trucano, J. Moya, G. Froehlich, A. Hodges, and D. Peercy. Guidelines for Sandia ASCI Verification and Validation Plans - Content and Format: Version 2.0. Technical Report SAND2000-3101, Sandia National Laboratories, 2000.

[29] S. Plimpton. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *Journal of Computational Physics*, 117, 1995.

[30] S. J. Plimpton, R. Pollock, and M. Stevens. Particle-Mesh Ewald and rRESPA for Parallel Molecular Dynamics Simulations. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, March 1997.

[31] S. J. Plimpton and A. P. Thompson. Computational Aspects of Many-body Potentials. *MRS Bulletin*, to appear May 2012.

[32] D.E. Post and L.G. Votta. Computational Science Demands a New Paradigm. *Physics Today*, 58(1):35–41, 2005.

[33] J.N. Reddy and D.K. Gartling. *The Finite Element Method in Heat Transfer and Fluid Dynamics*. CRC Press, 2nd edition, 2001.

[34] Y. Saad and M.H. Schultz. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM J. Sci. Stat. Comput.*, 7:856–869, 1986.

[35] H. Shan, N.J. Wright, J. Shalf, K. Yelick, M. Wagner, and N. Wichmann. A Preliminary Evaluation of the Hardware Acceleration of the Cray Gemini Interconnect for PGAS languages and comparison with MPI. In *Proceedings of the second international workshop on Performance modeling, benchmarking and simulation of high performance computing systems*, PMBS '11, pages 13–14, New York, NY, USA, 2011. ACM.

[36] H. Simon, T. Zacharia, and R. Stevens. Modeling and Simulation at the Exascale for Energy and the Environment: Report on the Advanced Scientific Computing Research Town Hall Meetings on Simulation and Modeling at the Exascale for Energy, Ecological Sustainability and Global Security (E3). Technical report, Office of Science, The U.S. Department of Energy, 2007.

[37] J.L. Tomkins et al. The Red Storm Architecture and Early Experiences with Multi-core Processors. *International Journal of Distributed Systems and Technologies (IJDST)*, 1(2), April – June 2010.

[38] T. G. Trucano, M. Pilch, and W.L. Oberkampf. General Concepts for Experimental Validation of ASCI Code Applications. Technical Report SAND2002-0341, Sandia National Laboratories, 2002.

[39] R. S. Tuminaro, M. Heroux, S. A. Hutchinson, and J. N. Shadid. Official aztec user's guide–version 2.1. Technical Report SAND99-8801J, Sandia National Laboratories, Albuquerque NM, 87185, Nov. 1999.

[40] L.G. Valiant. A Bridging Model for Parallel Computation. *Commun. ACM*, 33:103–111, August 1990.

[41] H. van der Vorst. Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13:631–644, 1992.

[42] C.T. Vaughan, M. Rajan, R.F. Barrett, D.W. Doerfler, and K.T. Pedretti. Investigating the Impact of the Cielo Cray XE6 Architecture on Scientific Application Codes. In *Workshop on Large Scale Parallel Processing, at the IEEE International Parallel & Distributed Processing Symposium (IPDPS) Meeting*, 2011. SAND 2010-8925C.

DISTRIBUTION:

1 Allen L. McPherson, Los Alamos National Laboratory, P.O. Box 1663, Los Alamos, NM 87545
1 Charles H. Still, Lawrence Livermore National Laboratory P.O. Box 808, Livermore, CA 94551-0808

1 MS 1319 James A. Ang, 1422
1 MS 1319 Richard F. Barrett, 1422
1 MS 0897 Teddy D. Blacker, 1543
1 MS 1322 Sudip S. Dosanjh, 1420
1 MS 1318 Robert J. Hoekstra, 1424
1 MS 1324 Robert W. Leland, 1400
1 MS 1323 William J. Rider, 1443
,
1 MS 0899 Technical Library, 9536 (electronic copy)

Sandia National Laboratories